What is Logo?

Molly Watt

Logo is a computer language which was developed to provide an environment which allows learning to take place as naturally as possible. Seymour Papert and his colleagues at Bolt Beranek and Newman and later at MIT set out to create a computer language which would combine the capabilities of artificial intelligence with the theories of Jean Piaget in order to allow a learner to build his own intellectual structures through estimation, interaction, experience and revision.

The Logo language is designed to provide an environment in which the child/learner is in charge of

- Setting a problem to solve.
- Making choices.
- Playing with the problem, experimenting and trying out solutions.
- Building on what he has already done to do something more.

The language is interactive. You learn it *at* the computer. After working with a problem for a while, you edit it, revise it, and then play your next steps.

With Logo, a young learner can enter directly into the world of turtle geometry. Without memorizing formulas, he can create procedures for drawing squares, triangles, and circles.

This is unlike my own experience in geometry. I learned geometry in high school only because I was required to take the course as preparation for college.

Geometric thinking is possible now for any learner without a series of prerequisites. At the Lamplighter School in Texas, three year olds have used Logo to explore turtle geometry. And MIT students use the concepts of turtle geometry used in Logo as a way to explore mathematics.

Seymour Papert is often quoted as saying that "Logo has no threshold, no ceiling." I have heard it said that Logo is a six year old's dream and a computer scientist's nightmare. Logo makes complex explorations possible for learners of all ages, without imposing artificial hurdles.

With Logo, a young learner can enter directly into the world of turtle geometry.

How does a child explore turtle geometry? It is really quite simple. Every child knows how to move from one place to another. Using his own experience in walking, a child "teaches" the turtle to move across a computer monitor leaving a trail or line to create a drawing.

Let's call our learner Wendy. Her first experience might be something like this:

1. She decides to experiment by drawing a square.
2. She paces a square on the floor to notice how she draws it.
3. She remembers how she did it and types a set of commands to the turtle which might look like this:

FORWARD 25 (no, not far enough)
FORWARD 25 (okay, that's enough)
RIGHT 40 (oh, that wasn't what I meant at all, let's try more) RIGHT 40, (well that's not it, try again) RIGHT 10 (that's it)
(let's see, what was it,) FORWARD 50
RIGHT 90
FORWARD 50
RIGHT 90

And so on until she has a square. With that accomplished, she can simplify the steps and teach the computer to do it by shifting to the edit mode. What shall the name of this procedure be?

Very often a student will name it with her own name, typing:

```
TO WENDY
FORWARD 50
RIGHT 90
FORWARD 50
RIGHT 90
FORWARD 50
RIGHT 90
FORWARD 50
END
```

The turtle will then "know" how to WENDY, and will execute a square every time the command WENDY is typed.

Now Wendy can enjoy playing with the procedure WENDY, and will probably type it over and over for the pleasure of watching the turtle "know" how to WENDY. The second WENDY will be drawn in a screen position which will surprise her, and our new Logo user will immediately type WENDY again just to see what happens. At the end of four

Molly Watt, Educational Alternatives, Gregg Lake Road, Antrim, NH 03440.

What is Logo, continued...

WENDYS, there is a new design which can be taught to the computer as a procedure containing the subprocedure WENDY. It looks a bit like a window, so ndy can type:

```
TO WINDOW
WENDY
WENDY
WENDY
WENDY
END
```

or

```
TO WINDOW
REPEAT 4 (WENDY)
END
```

Wendy has started the process of

So it is with this first turtle drawing: Suddenly the Logo user is part of the world which assigns airplane reservations, cashes checks, and launches space shots. This is a world of power, which has in this first lesson become part of Wendy's own learning continuum. The process is intrinsically motivating and fascinating.

As every person can learn to use a pencil, everyone can — and will — learn to program a computer.

Projects by New Users

I think that it is important to look at some examples of tasks that some real students have set for themselves as problems to solve when beginning to learn Logo and to use a computer.

My purpose is to demonstrate the many working styles and interests which Logo can support. These students were in my classes in Amherst and Cambridge, MA and were between the ages of 8 and 17.

After several class sessions in a hands-on laboratory, I often suggest to my students, "See if you can make the turtle draw your initials." Many try the project, seem to enjoy the challenge, and then go on to some other experiment.

Two students, Ted and Stella, were working side by side, and became completely absorbed in drawing letters. Each determined to create a complete alphabet.

Ted immediately wrote a procedure to produce the whole alphabet, before creat-

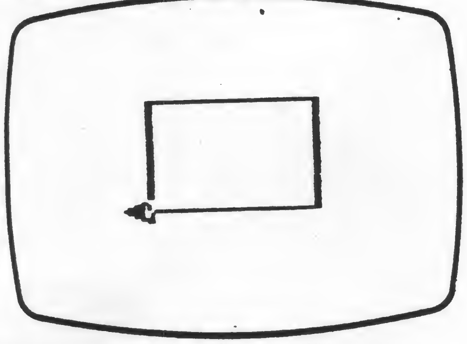


Figure 1. Wendy.

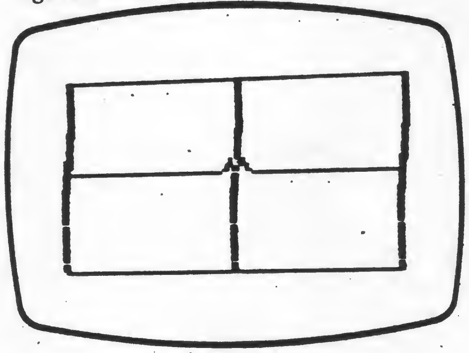


Figure 2. Window.

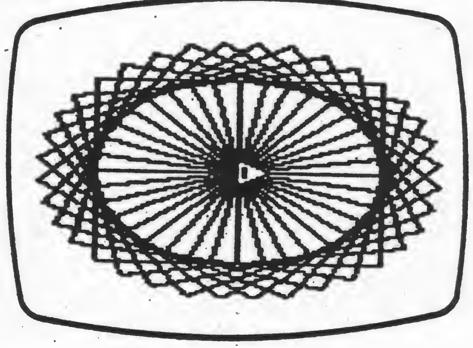


Figure 3. Rosewindow.

- earning to control the turtle.
- sing the computer as a tool.
- eaching the computer to execute an original idea and naming it.
- Noting the steps she used in order to save them as a procedure.
- Using the procedure as a subprocedure in her second procedure.

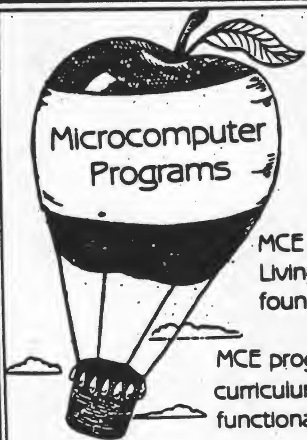
She has started to create her own computer language.

Taking the same example one step further, our new Logo user, in playing with WINDOW, may quickly discover that by using the procedure WINDOW and then rotating the turtle's position slightly a new design is made that holds a shape which begins to approximate a circle. This experiment can be formalized by typing, for example,

```
TO ROSEWINDOW
REPEAT 9 (WINDOW RIGHT 10)
END
```

As in all learning, the first job is to learn to control your tool. If you are using a pencil, you must practice holding it.

Your first written word may be your own name, which is a profound word to the writer. It permits you to show ownership, authorship, borrow library books, sign a check and send a greeting card rough the mail. It is still a long jump to riting a dissertation or even applying for a job. However that one word puts you into the powerful world of words which make up magazines, recipes and phone books.



Above the Crowd

MCE offers you superior microcomputer programs in Basic Living Skills. These programs have been field tested and found effective.

MCE programs are sophisticated yet simple to use. They offer curriculum flexibility in special ed., voc. ed., home economics, functional math and consumer ed. among others.

Eighteen programs are available from $44.95 for Apple II®. They feature: branching to individual reading and ability levels, a variety of instructional formats, vivid color graphics and advanced teacher control options.

To learn more about MCE programs call collect (616) 345-8681 or return the coupon below.

mce T.M.

MCE INC.
157 S. Kalamazoo Mall
Kalamazoo, MI 49007

MICROCOMPUTER EDUCATIONAL PROGRAMS

☐ Send MCE catalog ☐ Request demonstration

Name ______

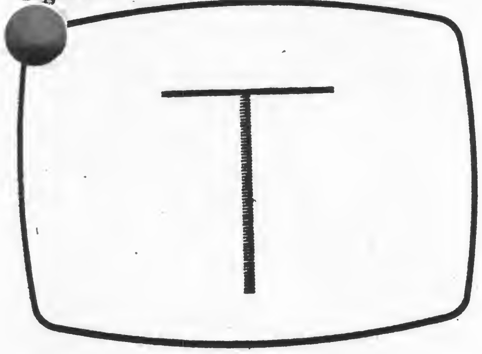
Address ______ H10

City ______ State ______ Zip ______

Phone ______ Best time ______

CIRCLE 199 ON READER SERVICE CARD

Figure 4. T.



Listing 1. Time.

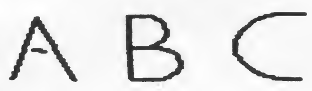
```
TO WAIT :T
 IF :T = 0 STOP
 WAIT :T - 1
END

TO TIME :HH :MM :SS
 TEST :SS = 60
 IFT MAKE "SS :SS - 60 TIME :HH ( :MM + 1 ) :SS
 IFF MAKE "SS :SS + 5
 TEST :MM = 60
 IFT MAKE "MM :MM - 60 TIME ( :HH + 1 ) :MM :SS
 TEST :HH = 12
 IFT MAKE "HH :HH - 11
 WAIT 200
 PRINT []
 ( PRINT :HH :MM :SS )
 TIME :HH :MM :SS
END

"FALSE is 1
"TRUE is 1
```

ing the individual letters. It looked like this:

```
TO ALPHABET
A
DRAW
B
DRAW
C
DRAW
.
.
.
END
```



Ted then began the long process of creating procedures to draw each letter in turn. After his alphabet was completed he was dissatisfied with the amount of time each letter was displayed on the screen. He began to work on the more complex programming problem of creating something for the computer to do invisibly so that the letter on the screen could continue to be displayed for a longer period. This was a natural extension of his alphabet procedure.

Stella started out to solve what appeared to be the same challenge as Ted's. During the process, she became fascinated by the proportions of height and width and how a variable input could change the relationships represented graphically.

The qualities of the computer which allowed her to experiment with inputs and then provided her with an almost instant reformulation of her letters heightened her excitement as well as her understanding about one way a computer could extend her thinking.

She began to collect patterns for monograms and initials which might make interesting designs on stationery. Her classmates used her program to think about embroidering on denim. For her final project, Stella created an art show using Logo procedures.

In one of the first Logo classes, Mark and Suzy became a working team. Mark seemed to need to be admired for his programming prowess and Suzy was reluctant to touch the computer.

This combination worried me because it seemed to reproduce the stereotyped, and I believe harmful, pattern of the way men and women relate to technology.

This twosome often seemed to have private jokes and to direct gestures toward classmates. Although I felt uncomfortable and wondered whether I should intervene, their classmates paid them no attention.

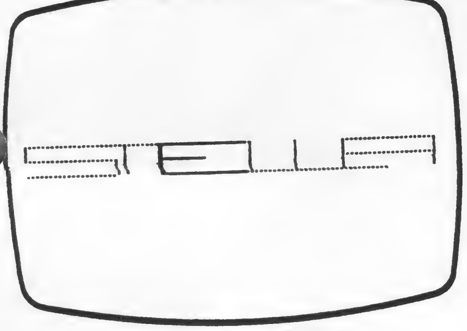
In the meantime, I watched and thought about them whenever I had a second to spare from this demanding programming class. I was aware that none of my suggestions had been accepted by either of them. While Mark was actively executing his ideas, Suzy admired, and I waited.

Then one day the duo split, each needing a computer to work on.

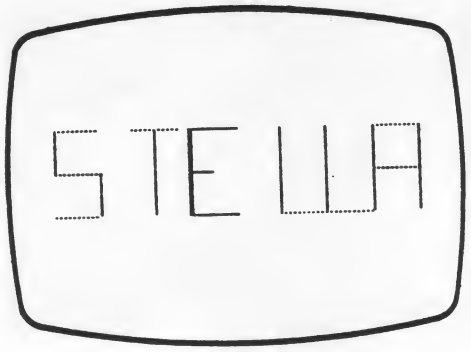
Mark was inventing a clock. It was a digital clock. The person using his program could type in the hour, minute and second and could then watch time move in five-second increments. His ability to capture some essence of time was satisfying to him, and truly mystifying to his classmates. Mark told them that his program was simply an approximation of time as is the time that all clocks keep.

Listing 1 shows how his program looked after a solid week of programming.

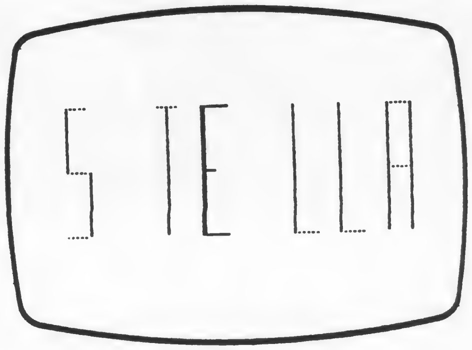
Figure 5.



S 10 50



S 25 35



S 70 7

His program does have some "bugs" which he has probably solved by now. One is the matter of conversion; the seconds do [illegible] convert to minutes after the numb[illegible]s reached. This was not discouraging [illegible] Mark. This was the next part of his program which needed attention in order to make it work better.

On the other side of the classroom, Suzy sat at a computer alone. At first she seemed to me to be immersed in a science fiction book. As I moved nearer, I saw that she was using the science fiction book as a reference for her work. She was working with an intensity I had not seen previously.

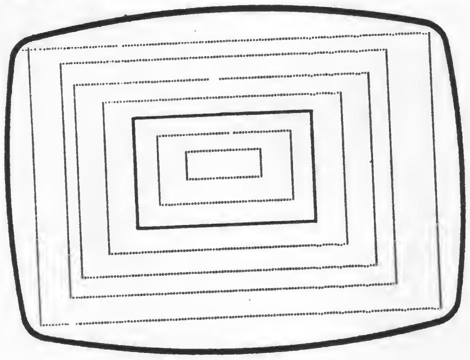
Between footstamps and fistshakes at the computer she was creating a science fiction book. She had discovered that the computer could print words. Linking her interest in reading science fiction with her limited knowledge of microcomputers, she had started on her first self-initiated project. She organized the procedures shown in Listing 2.

As the course ended, Suzy was beginning to experiment with turtle geometry to illustrate each chapter. My next steps for Suzy were to teach her how to save pictures drawn on the video screen, and to introduce her to the Logo text editor.

Suzy shared her project with her classmates during the last day of class. The science fiction aspect captured their [illegible]inations, and they demonstrated [illegible]ine interest in her work.

With our visions of what is possible for a Logo user to accomplish, Suzy's example may not seem dramatic. I include it because I believe we often push students to go too fast and forget the value of allowing a person to set his own goals. I do not believe that Suzy would have started sooner if I had structured the lessons differently; she needed a period of watching to get herself ready.

Figure 6. Door Through Time.



One visually oriented student spent many class periods creating drawings, saving his pictures directly on his disk without writing procedures. Jamie planned color and line relationships carefully and gave names which suggested to me the dimension of fantasy the computer offered him.

While classmates often gave pictures and procedures single letter names to avoid laborious typing, Jamie was content to take the time to type "Door Through Time," "Sparkle in the Night," "Experience in the Fifth Dimension," delighting classmates with his naming as much as with the pictures.

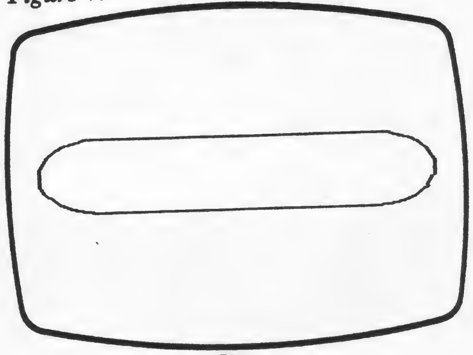
Another example of the way an individual entered the Logo environment or "mathland" is Jim. He simply started experimenting with the turtle.

He drew a shape which he later named "slot" because it looked like one. This slot with its circular ends and straight line connections intrigued him. During a series of lab periods, interrupted by several other projects, he progressed to "tslot" which added color and assumed a new position on the screen.

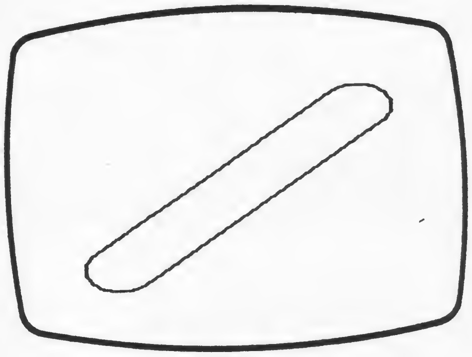
Next he played with variable inputs in order to experiment with size relationships. His moment of astonishment came when he discovered that the formula for a circle which he had memorized was approximate.

Jim was a serious student, and he often borrowed the Logo manual to read for homework. This provided him with many ideas about Logo primitives which might enable him to write more complex programs. He took himself beyond turtle

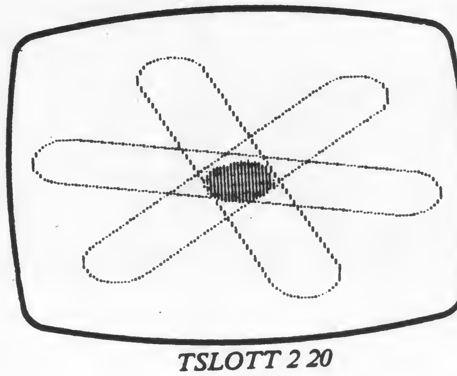
Figure 7.



Slot



TSlot



TSLOTT 2 20

Listing 2. Science Fiction.

```
TO MARS
 PRINT [A PRINCESS OF MARS]
 PRINT [THE GODS OF MARS]
 PRINT [THE WARLORD OF MARS]
 PRINT [THUVIA,MAID OF MARS]
 PRINT [THE CHESSMEN OF MARS]
 PRINT [THE MASTERMIND OF MARS]
 PRINT [A FIGHTING MAN OF MARS]
 PRINT [SWORDS OF MARS]
 PRINT [SYNTHETIC MEN OF MARS]
 PRINT [LLANA OF GATHOL]
 PRINT [JOHN CARTER OF MARS]
 PRINT [THIS IS THE MARS SERIES NO.1-11 IN ORDER
 BY EDGAR RICE BURROUGHS AUTHOR OF THE FAMEOUS
 TARZAN BOOKS]
 END

 TO DRAGONS
  PRINT [DRAGONFLIGHT]
  PRINT [DRAGONQUEST]
  PRINT [THE WHITE DRAGON]
  PRINT [THIS IS THE DRAGONRIDERS OF PERN TRILOGY
        BY ANN MCAFFERY]
 END
```

geometry to write an interactive program which enabled the user to choose a polygon ith any number of sides of user-specified ngth for the turtle to draw.

This project of Jim's pushed my programming ability, for I, too, was a new Logo user. I would often take my back-up copy of his disk home to work out various ways to solve his programming problems and bugs. After inventing or getting myself help with possible solutions or simpler examples, I created a new file called HELP to provide him with examples of possible solutions to his problem.

Jim, being an independent thinker, would study the examples in my HELP file, then say thoughtfully, "No, I don't believe that I want to do it that way," and continue with his own exploration and problem solving. I did notice that some of the HELP ideas found their way into his programming, and most of the inventions remained Jim's own.

At the end of the session, Jim was still working on centering his polygons above the text, and I had begun to use a HELP file with other students.

My first task was to invent a computer culture.

My last example is Manual. I have no file for his work. He was the class conceptualizer. He had started using computers because he enjoyed the fast action and energy of video games. He knew what would make an exciting program. Manual would strike up a conversation with a likely programmer describing an idea for a program. The talk continued as the other student moved toward a computer and started working out Manual's ideas. When Manual was satisfied that the person was "hooked" on the idea, he retreated and started another person on another idea.

I have Manual to thank for the work I do in the middle of the night, figuring out how to make a ball-shaped turtle appear to "bounce" off the edges of the screen.

These stories offer examples of the many ways in which students can explore a computer environment using the Logo Language.

Perhaps I've made it sound as if it "just happens." Because Logo permits a student to discover in a manner which is natural to the learning process it may appear that I have undervalued the teacher's role. Now we will look at some of the ways I have come to think about my role as teacher in a Logo classroom.

The Teacher's Role

Every computing class I have led has been made up of students with a wide variety of experience and knowledge about computers. Some students have never seen a real computer. Some may even be certain that the computer is *the* enemy of humanity and spend energy directing prejudicial comments toward it. Other students may own their own computers, belong to "user's clubs" and have been programming for years.

This situation in itself was somewhat unnerving to me. I had made a commitment to myself to become knowledgeable about Logo and instructional uses of computers in order to integrate this tool into a school curriculum. I was determined to create a relationship between the current computer revolution and life in school.

I had no models. I had never used a computer nor seen any person other than my husband use one. My first task was to invent a computer culture. This culture needed to be able to support all the students *and* me as a learner alongside them.

Any teacher's first job is to assess the situation and equip himself with:

- Knowledge
- Experience
- Hardware
- Time

We must examine our own attitudes about computers in our society, acknowledge the areas creating distress and identify the areas bringing optimism.

As Logo teachers our roles will cover a diverse, and perhaps uncomfortable range including demonstrator, teacher/lecturer, teller, time structurer, problem setter, management solver, arbitrator, decision maker, challenger, helper, collaborator, process sharer, question asker, idea extender, observer, documenter, admirer, enjoyer, time provider, technician, and model learner.

I require each student to keep a journal of process notes, questions and descriptions of problems encountered. I read these regularly and respond.

In response I might write a comment which is similar to an oral response, "That sounds frustrating,"; "Wow! You figured it out!"; "I'd like to see how your procedure works"; "Jim figured out how to do a similar problem, why not ask him for help next time?"; "I don't know the answer to this question, let's get together after school and visit the computer store."

The journals provide a valuable vehicle in which to keep track of progress and to allow patterns to become visible. Often simply describing a problem will allow a student to understand it more fully, and thus be able to solve it. Journals provide direct access to help. They enable students to formalize their own thinking.

The journals provide me with a sense of being in charge, of knowing what is going on, and, a means of keeping records of student work. They provide an opportunity for a personal relationship with each student on a daily basis. Since I am also learning Logo, the journals provide me with a sense of comfort; I can see what I need to learn, and decide what my own homework will be.

Beyond this, the journals provide both the students and me with an assurance that this is a collaborative learning experience, and that I am working *with* them. Confidence about this allieviates, I believe, for both the students and me the sense of anxiety which might otherwise be present in teaching and learning this subject. Most educators agree that anxiety interferes with learning.

As a Logo programming teacher, it is my job to make back-up copies of all my students' disks in order to protect the students from work loss due to damage or filing mistakes and to enable me to see the patterns in the work of individuals so I can plan my next formal lesson for the class.

Journals provide direct access to help. They enable students to formalize their own thinking.

I can collect small groups of students around a common interest or programming problem. Sometimes I will ask a student to share some work with the whole group as a teaching example or as a model. By examining the work on the back-up disks I can also determine whether a programming problem should be solved with a "gift" of a tool which the programmer is probably not yet ready to invent on his own. (Some tools I have given include procedures for creating circles, explosions, and countdowns.)

The disks give me time to work on programming problems by *trying* the program, and trying several solutions, away from the stress of a class period. Inevitably the disks force me to think about my own next learning steps.

During the actual lab time I wander, watch, listen, and answer.

I feel that the words I use are important.

Instead of solving a problem for a youngster by telling or showing the solution immediately, I usually say: "describe the problem," "Tell me what happens", "What did you want to have happen?"; or "Try it now and show me." Some teachers ask the student to "Teach me what you did."

This type of response is important for several reasons.

- It gives me, as teacher, time.
- It gives the student time.

- Description is a matter-of-fact task which can diminish emotion and allow the describer to see what actually happened clearly. (Frequently in the middle of describing, a student will say "Oh, never mind, I see what I did.")

If *after* a student has described the problem and neither of us knows how to solve it, we write a plan together in plain English words. It includes a statement describing what the student *wanted* to have happen. Usually when a solution is not clear to either of us, we are working with an example which is too difficult for us. We substitute a simpler problem "for practice."

Then, together we write a superprocedure in the same way that Ted wrote one for his alphabet, before he had invented the subprocedures. I make very sure that the first step in the procedure is one that the student and I can solve successfully right then.

Other jobs for the teacher include collecting, displaying and identifying resources.

I have rarely seen a computer which is not surrounded by a group interacting with it and kibbitzing with each other in a most congenial way.

I use bulletin boards to stimulate the learning process. I might post a weekly "Mystery Procedure," a new command with its definition and examples of its use, a challenging programming idea, a procedure to copy, a picture of a student's procedure, or a chart of students' names indicating their specific areas of expertise in order to make peer tutoring possible for every student. Sometimes I post an interactive program for students to copy, use, then modify and make their own.

Creating a Supportive Community

The computer community which we establish in our classrooms is for many students (and teachers) a first computer culture. As educators, we must be concerned about the values which are formed and used by this community of learners.

Many question my use of the word "community" in connection with computers. However, I have rarely seen a computer which is not surrounded by a group interacting with it and kibbitzing with each other in a most congenial way.

Most computers are a social *and* an intellectual center in a classroom unless usage is specifically regulated otherwise. Watchers see new ways to solve problems, participate in brainstorming new ideas and derive a great deal of pleasure from the process.

Teena Crowley, a third grade teacher, said, "I wish Logo were everyone's introduction to community work. Everyone is involved and offers input from the start."

Usually during a first class with a group I find it important to declare myself a Logo *learner* as well as a Logo teacher. I do it because it is honest and because it establishes a basis for collaboration. The responsibility for helping and teaching and creating this class belongs to all. It is an ideal opportunity for me to model the motto "Life Long Learning," certainly an important part of any curriculum.

Many issues emerge in the establishment of a Logo community. One of the first is the issue of ownership of procedures. Is it copying or stealing to save work which did not necessarily originate from the saver's own inspiration?

The nature of Logo filing and saving makes it almost impossible for beginning users to keep separate files, and so from the first time SAVE is typed, a file of mixed-up origin exists.

I prefer to be excited by the possibilities of this "sharing" rather than regret it as a necessary "evil." I legitimize the sharing, giving, changing, and using of one another's procedures as part of the context of the community we are creating together.

I share my work with my students and I encourage them to exchange procedures among themselves.

Classes soon begin to create a vocabulary specific to their community. In one class a procedure named WING — basically an outwardly spirally triangle which appeared to fly across the screen — became a favorite expression for a way of moving quickly across a space with arms rotating rhythmically. This reference point created a sense of cohesion and inclusion among classmates and added a dimension of good-natured humor.

I credit some of my success in creating a supportive community to my insistence that part of the experience in computer class is participating in discussions about ethical and responsible computer use.

I am genuinely worried about how our society makes decisions about computer use. I make clear to my students that computers are surrounded by a human culture with developed values and a sense of conscious choice about their use.

I believe that one way people develop into responsible, rational human beings is by participating in discussions about dilemmas with peers.

I often start a discussion with a simple story. Ideally, it describes an issue which is real or potential issue for this class, without embarrassing any group member. Once I told about a filing mistake I made when making a back-up copy of Ted's alphabet. Half the procedures for creating letters disappeared through my mistake. I had to tell him what I had done to his work, and how sorry I was to have made such a silly and harmful mistake.

Part of the experience in computer class is participating in discussions about ethical and responsible computer use.

A 20-minute period of sharing of mistake stories followed. No one was required to talk, but all class members were expected to participate by being part of our circle. The rules also prevented comments or judgments being directed towards any person or point of view. Discussion was encouraged.

Other issues which have worked for discussions in my classes are:

- Scheduling of computer use.
- What about someone who doesn't take a turn?
- Is it okay to borrow disks, copy disks, change disks?
- What about using "bad" or "dirty" words to name procedures?
- Should all schools have computers?
- Should all kids have to learn to program a computer?
- Why would a girl want to use a computer?
- What kinds of information should a school computer keep?
- Is it possible to have a really "private" computer file?
- What are some ways computers work in our lives?
- How can I get my family to value my programming instead of dismissing me as a "brain"?

The discussion which touched me the most deeply was requested by my students at the end of a summer school session. The subject was: now that we know so much about computer programming, how can we help our teachers feel comfortable knowing less than we do, so we can have computers in schools?

Beyond these ways of thinking about structuring a class and a curriculum, there are many opportunities for collaborative programming. One teacher has helped her students create a class adventure game using Logo.

The class planned the rooms, and then pairs or individuals created the procedures to make each room. Her role was to set up the structure and aid in the process of linking.

Harold Abelson's manual for Logo gives many examples of interactive programs which can be enjoyed by a class. Many teachers are using it to create their own programs for guessing numbers and creating crazy sentences.

Dan Watt's work creating curriculum for a dynamic turtle which moves according to laws of physics is interesting to use and then modify.

A great need exists for all Logo teachers to share their emerging curricula. This will enhance our collective understanding and provide more models about the ways students can work within a Logo environment.

What Logo Teachers Say They Teach

- Computer literacy.
- The history and learning theory in Logo.
- How to program a computer-like character, a turtle robot or Bit Trak.
- Controlling a turtle on a screen.
- How to pace out shapes and then teach the turtle to draw shapes.
- How to change pencolors and background colors.
- How to edit.
- How to initialize a disk.
- How to draw initials.
- How to use repeat.
- How to use subprocedures in procedures.
- How to use recursion.
- How to read a print-out of programs.
- How to draw procedure trees.
- How to use variables.
- How to use the Logo Manual.
- How to manage files and clear the workplace.
- How to use existing interactive programs and modify them.

This simple and straightforward list may be more or less what you expected. However when I ask my Logo students, whether school children or professional educators, what they learned in Logo class the list is quite different — fuller and more profound.

What Logo Students Say They Learn

- About problem solving and estimation.
- About thinking and learning styles.

This person used to develop applications for the 8086/8088 market with the best of them.

SOFTech MICROSYSTEMS

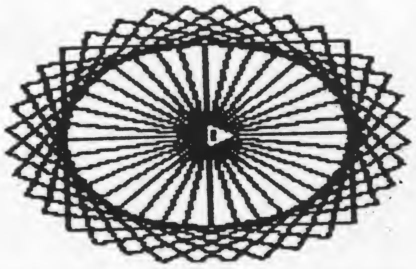
- About how to use their own learning style.
- To think logically.
- To work without emotional manipulations; the computer doesn't care whether you feel angry.
- To use procedural thinking.
- To use strategies for problem solving.
- To become comfortable thinking mathematically.
- To be able to think geometrically.
- To be able to consider laws of motion.
- About language by creating my own system for naming procedures.
- About graphics and design.
- How important revision of procedures and text is, and how simple it is to do.
- That decimals are useful.
- How to type.
- How to be patient.
- How to take risks in working.

The most significant things they claim to have learned include:

- Looking at their own mistakes with an interest in understanding what happened instead of shame.
- Feeling competent in setting their own problems and supported in solving them.
- Understanding that learning and doing involve frustration and ease; they go quickly or slowly, parts are intriguing or boring, and this is what makes up all work and life.

There is no question in my mind that working with computers is one way of forming a direct link with a sense of the future. Computers can provide a sense of optimism, new frontiers to be explored, and a kingdom to be conquered.

Logo was developed to create an interactive environment, a mathland, in which students could set their own pace, problems and goals. It is a comfortable way for me to enter the future. It is a challenging way and it is a way that I can understand, choose and control. □



This person used to develop applications for the Z-80 market with the best of them.

SOFTECH MICROSYSTEMS

Why Logo?

Logo is designed to encourage development of problem-solving skills.

Brian Harvey
Logo Computer Systems Inc.
368 Congress St.
Boston, MA 02210

Logo is a language for learning. That sentence, one of the slogans of the Logo movement, contains a subtle pun. The obvious meaning is that Logo is a language for learning programming; it is designed to make computer programming as easy as p[illegible]ble to understand. But Logo is a[illegible] language for learning in general. To put it somewhat grandly, Logo is a language for learning how to think. Its history is rooted strongly in computer-science research, especially in artificial intelligence. But it is also rooted in Jean Piaget's research into how children develop thinking skills.

In a certain sense, all programming languages are the same. That is, if you can solve a problem in one language, you can solve it in another—somehow. What makes languages different is that some types of problems are easier to solve in one language than in another. Language designers decide what kinds of problems their language should do best. They then make design choices in terms of those goals.

[illegible]t the Author
[illegible]n Harvey is director of the computer center at Lincoln-Sudbury Regional High School, 390 Lincoln Rd., Sudbury, MA 01776. He modified Logo for its PDP-11 minicomputer.

Logo as a Programming Language

Let's postpone for a while the broader educational issues. First, we'll consider Logo simply as a programming language. How is it similar to other languages; how is it different? Syntactic details aside, there are several substantial points of language design through which Logo can be compared to other languages.

Logo is procedural. A programming project in Logo is not written as one huge program. Instead, the problem is divided into small pieces, and a separate *procedure* is written for each piece. In this respect, Logo is like most modern languages. Pascal, APL, LISP, C, and even FORTRAN permit the division of a program into independent procedures. Among the popular general-purpose languages, only BASIC lacks this capability. (The sample Logo programs in this article are written in Apple Logo, a dialect written by Logo Computer Systems Inc. Other versions of Logo will be slightly different in details.)

Consider the Logo program in listing 1a. Even if you don't know anything about Logo, it's probably obvious what this pair of procedures does. Compare it to the BASIC version in listing 1b.

The GOSUB construct in BASIC is weaker than a true procedure capability in several ways. For one thing, the BASIC subroutine is not an independent program; if line 100 were omitted, the program would "fall into" the subroutine. More important, there is no concept in BASIC of inputs to procedures, like QUESTION and ANSWER in the Logo program. Instead, extra statements must be used to assign values to the variables Q$ and A$, explicitly.

This explicit assignment is not simply an inconvenience. It means that the main part of the program has to "know" about the inner workings of the subroutine. In the Logo version, the procedure named QUIZ knows only that the procedure QA has two inputs, a question and an answer. If QA were modified to use different names for the variables, QUIZ would still work. Similarly, although this particular example doesn't show it, Logo procedures can have an *output* that is communicated to the calling procedure. (The DEF statement in BASIC provides a limited version of procedures with outputs; the limitations are that the inputs and outputs must be numbers, and the definition must be a single line without conditional branching.)

Logo is interactive. Like BASIC, but unlike Pascal, Logo lets you type in a command to be carried out *right away*. It's also quick and easy to change one line of a program. Other interactive languages are LISP and

Listing 1: *Comparison of Logo and BASIC. Each program asks the same set of three questions and compares the user's response to the author's answer. In the BASIC version (listing 1b), the "questioning" subroutine (lines 1000–1110) is not an independent program. In the Logo version (listing 1a), the procedure QA could stand alone, and might conceivably be used by other programs.*

(1a)

```
TO QUIZ
QA [WHAT'S THE BEST MOVIE EVER?] [CASABLANCA]
QA [HOW MUCH IS 2+2?] [5]
QA [WHO WROTE "COMPULSORY MISEDUCATION"?] [PAUL GOODMAN]
END

TO QA :QUESTION :ANSWER
TYPE :QUESTION
TEST EQUALP :ANSWER READLIST
IFTRUE [PRINT [YOU'RE RIGHT!]]
IFFALSE [PRINT SENTENCE [NO, DUMMY, IT'S] :ANSWER]
END
```

(1b)

```
10 Q$="WHAT'S THE BEST MOVIE EVER?"
20 A$="CASABLANCA"
30 GOSUB 1000
40 Q$="HOW MUCH IS 2+2?"
50 A$="5"
60 GOSUB 1000
70 Q$="WHO WROTE 'COMPULSORY MISEDUCATION'?"
80 A$="PAUL GOODMAN"
90 GOSUB 1000
100 GOTO 9999
1000 PRINT Q$;
1010 INPUT R$
1020 IF R$=A$ THEN GOTO 1100
1030 PRINT "NO, DUMMY, IT'S ";A$
1040 RETURN
1100 PRINT "YOU'RE RIGHT!"
1110 RETURN
9999 END
```

APL; other noninteractive languages are C and FORTRAN.

Whether or not a language is interactive has an effect on its efficiency. In brief, program development is generally faster with an interactive language, but already-written programs generally run faster in a language that is not interactive. The difference has to do with the mechanism by which the computer "understands" your program.

Every computer is built to understand one particular language. This machine language is different for each type of computer. Since machine-language instructions are represented as numbers, they're not easy for people to read. For example, the number 23147265 might mean "add the number in memory location number 147 to the number in memory location 265." Programs written in a high-level language, including Logo and the other languages mentioned here, must be translated into machine language before the computer can carry them out. This translation is done by another computer program that comes in one of two flavors: compiler or interpreter.

A Pascal compiler, for example, takes a program written in Pascal and translates (compiles) it into the machine language of whatever computer you're using. The translated program is permanently saved as machine language (probably as a file on your floppy disk). Thereafter, the machine-language program can be executed directly. The compiling process takes a long time. But once it's finished, running the compiled program is very fast because it need never be compiled again.

A Logo interpreter, on the other hand, does not create a permanent machine-language version of your program. Instead, each Logo statement is translated and executed every time the statement is supposed to be executed. The interpreter does not produce a machine-language representation of your program, but simply carries out the machine-language steps itself. If a Logo statement is to be executed six times, it's translated six times. (Actually, some interpreters, including Apple Logo, save a partial translation of each procedure, so that the second execution is somewhat faster than the first; this process is too complicated to explain in this article.)

Interpreted languages can be interactive. Suppose you want to find the value of 2+2 in Pascal. First, you must use the text-editor part of your Pascal system to write a disk file containing a Pascal program. Then, you run the Pascal compiler, which will translate the program into machine language. Finally, you run the compiled program and your computer types out 4. In an interpreted language like Logo, you can simply type PRINT 2+2 to see the same result.

The situation in which interaction is most important is program development. If you are writing a complicated program, it probably won't work right the first time you try it. You'll have to try it, see what goes wrong, change the program, and try again. In order to see what went wrong, you'd like to be able to use interactive debugging. (You stop the program where the error happens and type in commands to examine the values of variables at that moment.) This debugging cycle may be repeated many times before the program finally works completely. Even though a compiler might make the program run faster, an interpreter is likely to make the entire debugging process faster because it's so much easier to find and fix your mistakes. It's only after the program works, and you want to use it every day without modification, that the compiled version is really faster.

The flexibility and ease of use of an interactive language is particularly valuable in an educational setting. For a student of programming, there often is no production phase—the program is of interest only as long as it doesn't work. When it does work, the student goes on to the next problem. In that sort of environment, the speed advantage of the compiler never materializes. In a business environment, on the other hand, the actual production use of a program is likely to be more important, which makes a compiler more desirable.

Some languages use mixed schemes. BASIC (normally an interpreted language) has compilers that allow the user to give up interaction for efficiency. Some LISP compilers can coexist with interpreters, so that some procedures can be compiled while others are being debugged interactively. Some versions of Pascal are compiled into an intermediate language called p-code, which is then interpreted. FORTH uses a similar system of partial compilation, but the compiler is part of the run-time environment, so single statements can be compiled and run interactively.

Logo is recursive. In a procedural language, one procedure can use another procedure as a *subprocedure* to do part of its work. A language is *recursive* if a procedure can be a subprocedure of itself.

All modern procedural languages allow recursion. Among widely used languages, only FORTRAN allows procedures but not recursion. (BASIC, as was mentioned earlier, has neither.) It may seem as though recursion isn't too important. Why should it be any different from any other use of subprocedures? It's hard to explain in a simple way why recursion is important. The idea behind recursion, though, has profound mathematical importance. By allowing a complicated problem to be described in terms of simpler versions of itself, recursion allows very large problems to be stated in a very compact form.

A well-known example of a problem best solved using recursion is the Tower of Hanoi puzzle. This puzzle

Orange Micro
PRINTER STORE

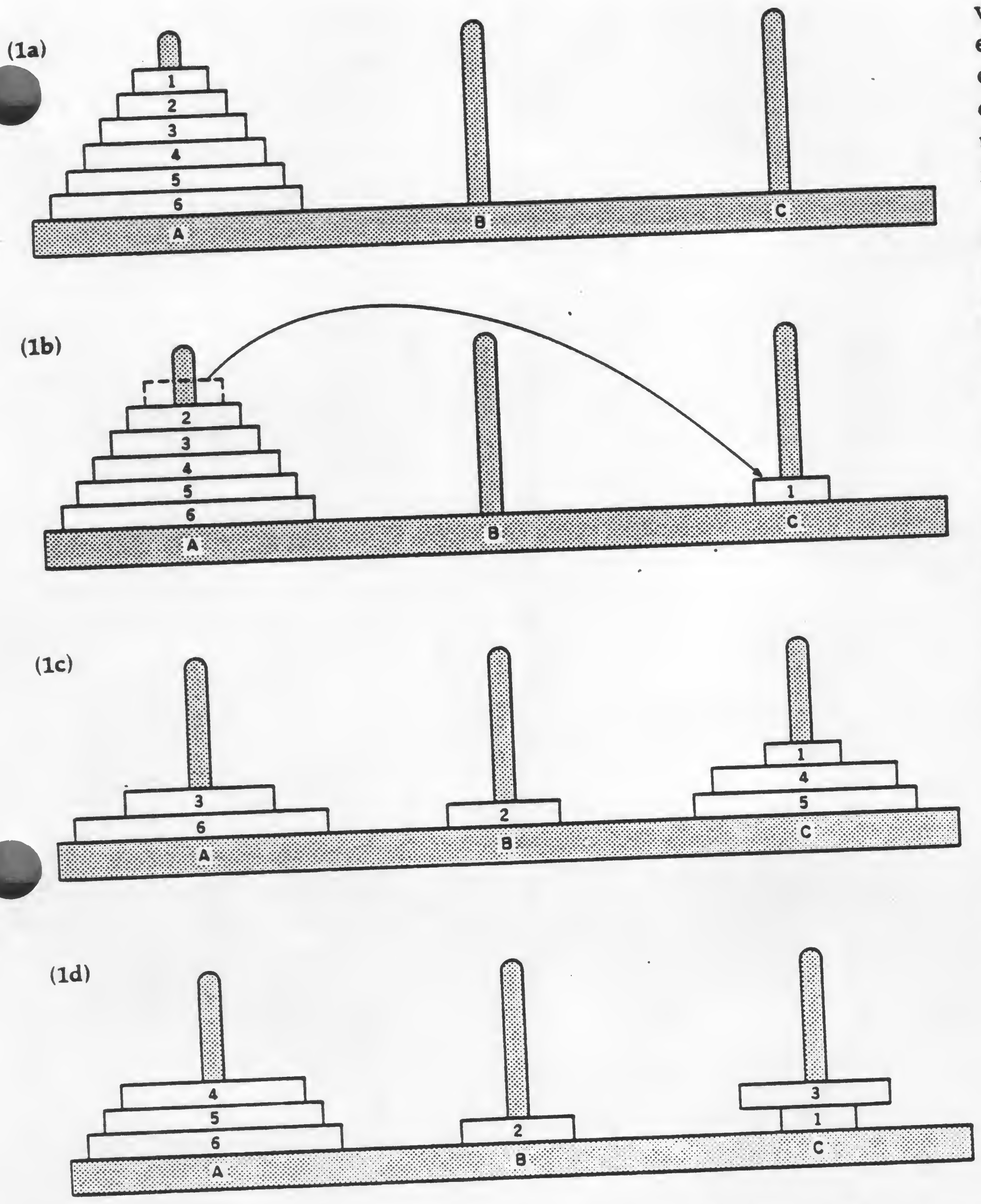


Figure 1: *Typical moves in the Tower of Hanoi puzzle. Figure 1a shows the initial position, figure 1b the first move, and figure 1c the position after several moves. Figure 1d shows an illegal situation with a larger disk on a smaller one.*

has a number of different-size disks piled initially on one of three pegs, with the smallest at the top. The problem is to move the disks onto a different peg, moving one disk at a time and never moving a disk onto a smaller disk (see figure 1).

To solve this problem, first notice that it's very easy with only two disks (see figure 2a). It's easy to see that we have to get disk 2 onto peg B somehow. To do that, we have to get disk 1 out of the way. Therefore, move disk 1 to peg C, disk 2 to peg B, and disk 1 to peg B.

Now suppose there are six disks (see figure 2b). Again, we have to begin by getting disk 6, the largest one, from peg A to peg B. But now there are five disks in the way, not just one. This provides us with a subproblem: move five disks from peg A to peg C. But this is exactly the Tower of Hanoi puzzle itself with five disks instead of six! The subproblem is a simpler version of the main problem. This calls for the recursive solution shown in listing 2.

In working through this program, bear in mind that each use of the HANOI procedure has its own, private variables; the value of NUMBER, for example, remains constant throughout any particular use of the procedure, even though there is another use of HANOI with a different value for NUMBER in the middle.

In addition to Logo, many other languages allow recursion (these include Pascal, C, LISP, and APL). The style of Logo, however, encourages the use of recursion more than some other languages. C and Pascal allow recursion but encourage iteration. (Iteration means telling the computer to execute something repeatedly. The FOR. . .NEXT construct in BASIC is an example.) Logo is the other way around: iteration is possible, but recursion is preferred. For many purposes, neither approach is clearly right. Iteration is somewhat simpler for the situations in which it works at all; in some cases like the Tower of Hanoi puzzle, however, nothing but recursion will do.

Until recently, iteration was much more efficient than recursion, both in speed and in the use of memory. A major advance in recent implementations of Logo, including the versions available for the Apple II and the Texas Instruments TI-99/4A microcomputers, is that *tail recursion* is recognized by the interpreter and treated as if it were written as iteration. Tail recursion is the situation in which the recursive use of a procedure is the last thing done in the procedure. In general, it is only tail-recursive programs that could just as easily be done iteratively. The HANOI procedure, for example, is not tail recursive because two recursive procedure calls are in it, only one of which is at the end.

Logo has list processing. Every major programming language has some way to group several pieces of information (numbers, for example) into one large unit. In FORTRAN and BASIC, this mechanism is the array. In Pascal and C, arrays are also used, along with a more complicated grouping called a record in Pascal or a structure in C. In Logo, the main grouping mechanism is called the *list*.

Lists and arrays have two major differences. First, arrays have a fixed

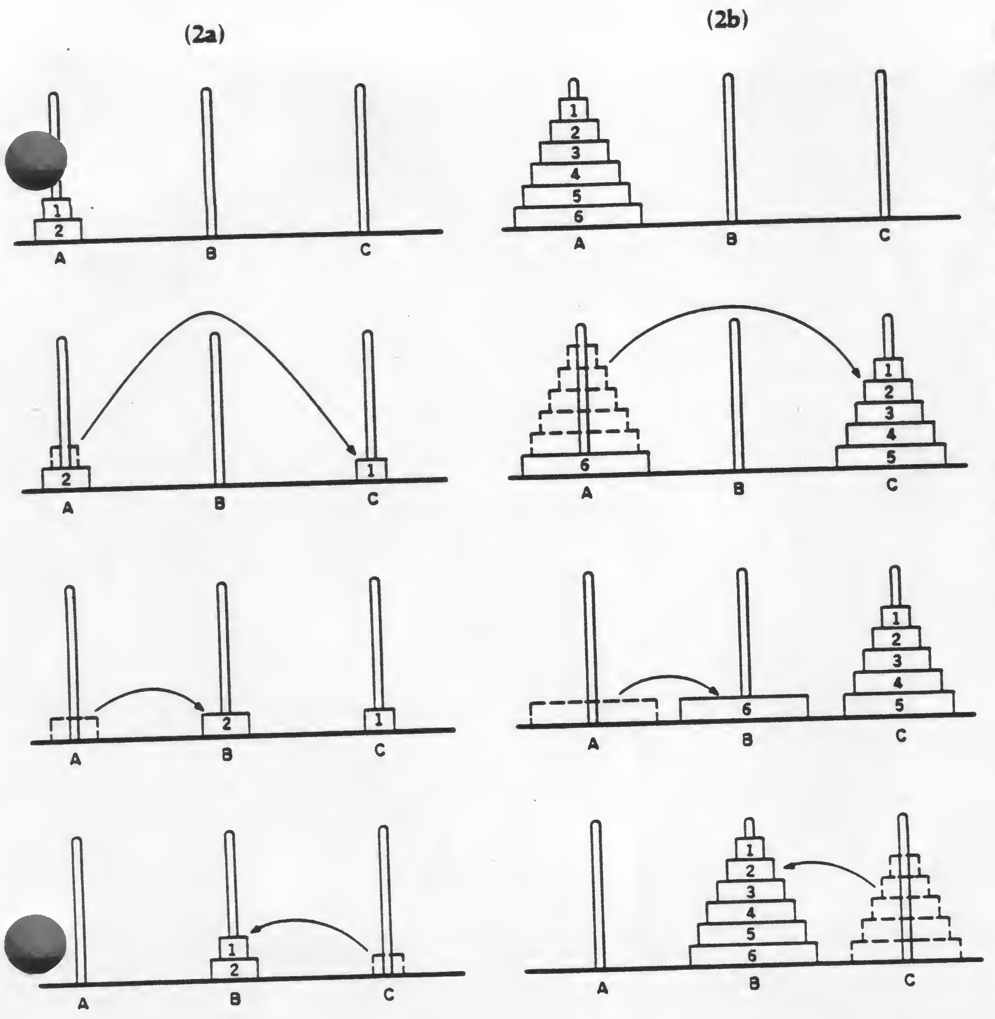


Figure 2: *How the puzzle breaks down into simpler subproblems with similar solutions. Figure 2a shows the simplest solution to a puzzle involving only two disks. Figure 2b shows the situation when the same procedure is used on more disks.*

Listing 2: *General solution to the Tower of Hanoi puzzle in Logo. The program requires four inputs. The variable* NUMBER *tells the program how many disks to the puzzle; the other three inputs are the names of the pegs. The* IF *statement detects the trivial subproblem of moving zero disks, for which there is nothing to do.*

The solution is found by dividing the problem into a series of simpler subproblems, all of which can be solved by repeating a simple series of moves. First, move all but the bottom disk to the third peg; then, move the bottom disk to the destination peg; and finally, move all but the bottom disk to the destination peg (see figure 2).

```
TO HANOI :NUMBER :FROM :TO :OTHER
IF :NUMBER = 0 [STOP]
HANOI :NUMBER – 1 :FROM :OTHER :TO
PRINT (SENTENCE [MOVE DISK] :NUMBER [FROM PEG] :FROM [TO PEG] :TO)
HANOI :NUMBER – 1 :OTHER :TO :FROM
END

HANOI 6 "A "B "C
```

size, while lists can become bigger or smaller as a program executes. (There is no equivalent in Logo to BASIC's DIM statement, which is used to specify how big an array will be.) The second difference is that arrays must be uniform. That is, you can have an array of 12 numbers, or an array of strings each 23 characters long, but you can't have an array of some of both. (A Pascal record or C structure can have some of both, but only in one predeclared pattern.) Each element of a Logo list can be any Logo object: a number, a word, or even another list. Thus, the following are examples of lists:

```
[VANILLA CHOCOLATE MOCHA]

[VANILLA [MINT CHOCOLATE CHIP]
  [FUDGE SWIRL]]

[BANANA 3.14159 [RED BLUE
  YELLOW] 2.71828]

[FLAVORS [VANILLA CHOCOLATE]
  SIZES [LARGE SMALL]
  OPTIONS [[HOT FUDGE]
  [SUGAR CONE]]]
```

The first of these is a list of three words. The second is also a list with three members, but the first is a word and the others are lists. The third example shows that numbers can be included. The last example demonstrates that a list can contain a list that contains a list.

The last example is a special kind of list, called a property list. If this property list were associated with the name ICECREAM, the Logo statement

```
PRINT GPROP "ICECREAM "SIZES
```

would print:

LARGE SMALL

(GPROP stands for Get PROPerty.) Property lists are a convenient way to group related information. Imagine, for example, a Spacewar game program with several ships, each with a property list. The properties might be the ship's position, velocity, shape, remaining energy, and so on.

The reason that some languages restrict you to using arrays is that, being uniform and of fixed size, they are more efficient to deal with. The restrictions on arrays mean that if the computer knows where the beginning of some array is located in memory, the location of the *n*th element of the array can be calculated easily, no matter what values the elements actually have.

With a list, the size of each element is variable. Therefore, lists are stored in a more complicated way. As a result, to find the fourteenth element, you have to start with the first one, figure out where the second one is, then figure out where the third one is, etc. Since this is all done automatically by the Logo interpreter, lists aren't hard for the programmer to use, but it's somewhat slower than finding something inside an array.

Among major languages, LISP uses lists much like those in Logo. (In fact, the data structures in Logo are based on those of LISP. LISP's name stands for LISt Processing.) APL uses a data structure that is like lists in that it is not fixed in size, but is like arrays in that it is uniform in composition. In other words, an APL vector can grow or shrink, but it has to be all numbers or all characters. Pascal and C don't have lists, but they have pointer variables that can be used along with records or structures to build the equivalent of lists. FORTRAN and BASIC don't have dynamic storage allocation—you can't make something bigger in the middle of the program—so there is no way to create lists in them.

Logo is not typed. In BASIC, if you want a variable to contain a character string, you put a dollar sign at the end of its name. If you don't use the dollar sign, the variable must contain a number, not a string. (Some versions of BASIC have a third type: a variable whose name ends with a percent sign contains an integer, or whole number.) In Pascal and C, the *type* of a variable must be given explicitly in a declaration. In FORTRAN, variables can be declared as in Pascal; if a variable isn't declared, its type depends on the first letter of its name. The letters I through N indicate integer variables.

In Logo, as in LISP and APL, variables are not typed. Any variable can take on any value. The same variable can be an integer at one point in the program and a character string (called a word in Logo) later on.

Originally, variable typing wasn't a matter of language-design philosophy. Variables were typed to make life easier for the people who wrote compilers. Since different machine-language instructions are used, for example, to add integers and to add numbers with fractional parts, it's easier to translate "A+B" into machine language if you know ahead of time whether or not A and B are integers.

More recently, some language designers have taken the position that variable typing is a good thing, apart from implementation issues, because it disciplines the programmer to use a variable for only one purpose. In re-

Introducing the Honorable Shell—

A true breakthrough in operational technology for CP/M™ systems.
OKARA lets you, the user, create command scripts to replace the boring, repetitive, error-prone keyboard dialogs normally required by CP/M and the programs that run under it. With OKARA the machine presents any personality you design, from simple cryptic commands that do it all to elaborate, hand-holding menus. OKARA is a must for anyone who uses CP/M. And a super-must for system integrators, software developers, office managers, teachers, parents... anyone who sets up CP/M applications for others.

☐ **Fast**—20 to 100 times faster than SUBMIT, SUPERVYZ™ or STOK PILOT.™

☐ **Tiny**—Typically under 3K with all modules loaded, plus 2K for CCP.

☐ **Smart**—You can use it like SUBMIT, or program it to create your own menus or other dialogs.

☐ **Simple**—If you use CP/M, you can learn OKARA in an hour or less.

☐ **Slick**—Provides some enhancements that CP/M has always needed.

☐ **Friendly**—In so many ways. One example, OKARA can page console output from any program or command. No more racing to type a control-S.

☐ **Universal**—Nearly all present and future CP/M applications will run under OKARA without modification.

☐ **Price**—$150, single copy. $130, disk only. $20, manual only. OEM discounts available.

☐ **Formats**—8" single density, 5" Osborne, Apple, Northstar.

☐ **System**—8080, Z80, or 8085 cpu, CP/M 2.2 Operating System. Text editor to create scripts.

P.O. Box 11461
Oakland, CA 94611
(415) 654-8671

*OKARA is a trademark of KIAI Systems. SUPERVYZ is a trademark of EPIC Computer Corporation. STOK PILOT is a trademark of STOK Computer Interface. CP/M is a trademark of Digital Research.

Circle 223 on inquiry card.

Listing 3: *Logo variables are nontyped. The variable* NUMBERS *contains whatever the user enters. First, it is examined as a list of words and tested to see if it contains the value* DONE; *next, it is used as a list of numbers and added.*

```
O ADDLOOP
RINT [TYPE TWO NUMBERS TO ADD.]
MAKE "NUMBERS READLIST
IF FIRST :NUMBERS = "DONE [STOP]
PRINT SENTENCE [THE SUM IS] (FIRST :NUMBERS) + (LAST :NUMBERS)
ADDLOOP
END
```

jecting typing, the designers of Logo did not mean to encourage the haphazard use of variables for different purposes; rather, they built a procedural language in which variables are attached to a particular procedure, rather than being available to the entire program. This encourages the same discipline in a different way.

As an example in which typed variables are awkward to use, listing 3 illustrates the common problem of writing a program that reads some numbers entered by the user, performs some calculation with them, and repeats the process until the user signals that there are no more problems to do.

This program has been written so that the user can enter the word DONE when no more numbers are left to add. In a typed language, the numbers would have to be read into a numeric-type variable, not a string-type variable. Entering a nonnumeric word would be an error. FORTRAN programs used to be full of instructions to the user like "type 9999 to indicate that you're done." Pascal programs face the same difficulty.

Logo is extensible. Every computer language has certain built-in, or primitive, operations. Most languages, for example, include arithmetic operations on numbers, and some way to print the results. Procedural languages allow the programmer to create new operations, extending the capability of the language. In that sense, most languages are extensible. But "extensible" is used by language designers in a special sense.

An *extensible* language is one in which user-defined procedures "look like" primitive procedures. This is partly a matter of notation and partly a matter of real power. In most languages, the primitive arithmetic operations can be applied to several different types of variables (integer and real, for example) with appropriate results for each type. In most languages, however, user-defined procedures must specify in their definition one particular type of variable to which they apply. This restriction violates the principle of extensibility.

Extensible languages are particularly valuable for teaching because a teacher can provide language extensions and teach them as if they were primitives. LISP, Logo, APL, and FORTH are extensible, with some minor restrictions in some cases. Logo violates pure extensibility, for example, in that some of the primitive arithmetic operations are represented in infix form (with the operation symbol between the two operands, as in 3 + 2), while user-defined procedures can be represented only in prefix form (with the operation symbol before the operands, as in SUM 3 2). Almost all Logo primitives are used in prefix form.

As an example of the use of extensibility in Logo, most versions do not have primitive procedures for iterative looping, like the FOR, DO, or WHILE constructs in other languages. But it is very easy to define these procedures, if you want them, so that they look syntactically similar to the IF command that is a Logo primitive.

Logo as a Learning Language

Among respectable languages, you may have noticed two groupings.

WICAT

has the 68000

While others are promising, WICAT Systems for nearly two years has been quietly delivering their complete 68000 computers.

The System 150 is a popular desktop computer with large memory and hard disk. unmatched in price/performance. The System 100 is its expandable software compatible high performance companion.

Concurrent s **integral™ database** and **menusystem** greatly accelerate the development of applications. Full UNIX™ System 3 with no compromises is ready

Concurrent can discuss your needs and help you select the proper configuration Please write or call (513) 281-1270

Motorola 68000. 256K memory. 10MB Winchester disk. 760KB floppy. terminal. system software. choice of Pascal. FORTRAN. C. Assembler BASIC. COBOL

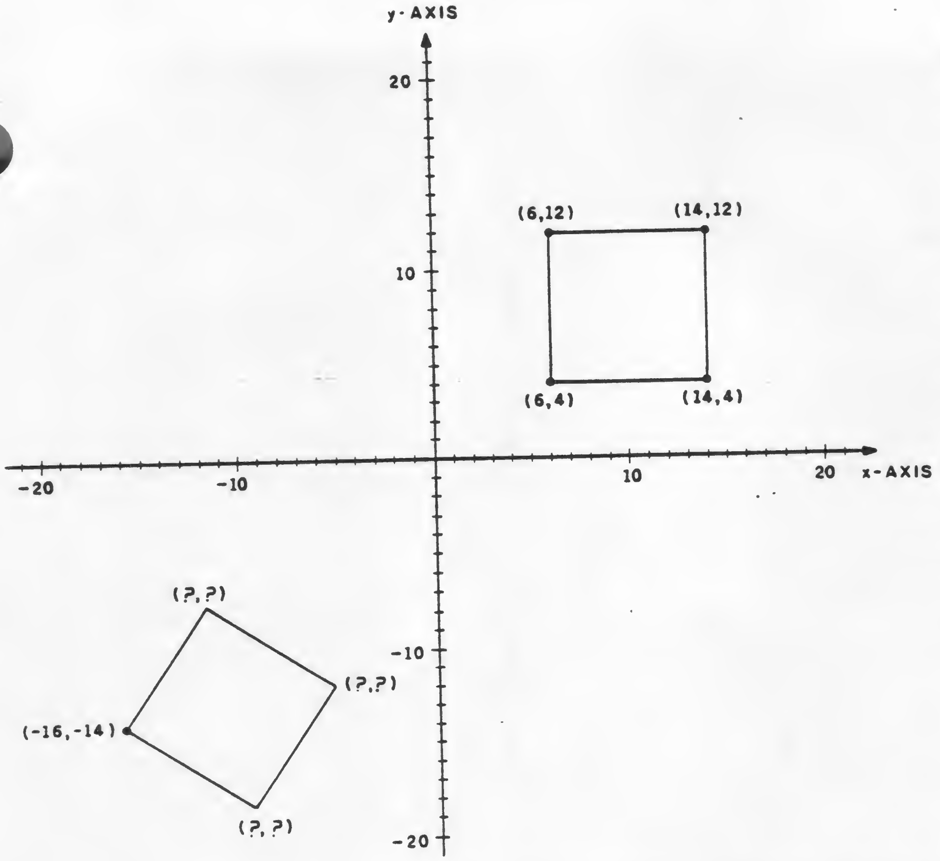
Single user 150-1	$ 9,450
Three users 150-3	$10,850
Six users 150-6(512K) .	$12,850

MULTIBUS is a trademark of INTEL Corporation
UNIX is a trademark of Bell Laboratories

OPTIONS graphics printers. IEEE 488 instrumentations interface videodisk controllers A/D boards

Concurrent Corporation
1870 Madison Road Cincinnati, Ohio 45206

Circle 112 on inquiry card.



igure 3: *The difficulties involved in graphics using Cartesian coordinates. A square is imple to draw when its sides are parallel to the axes, but trigonometry is necessary when other orientations are used.*

Logo, LISP, and APL are interpreted, list-oriented, and untyped. Pascal and C are compiled, array-oriented, and typed. (All respectable languages are procedural, by definition.) These groupings reflect historical accidents, implementation convenience, and language-design philosophy. For example, C and Pascal are very similar because they are both derived from an earlier language, ALGOL, that established a style followed by many newer languages.

Compilers have a much easier time with typed languages, while interpreters are just as happy with untyped ones. The list-oriented languages were all invented by people who are primarily mathematicians, rather than computer programmers.

Within each group, though, the differences tend to reflect the particular use each designer had in mind. For example, C is different from Pascal largely because C was designed as a language for systems programming. In the list-oriented group, LISP was developed for use in artificial-intelligence research, and APL was developed to teach algebra and the mathematical topics, like calculus, that depend on algebra. Logo, though, was developed as a learning language, not for a specific branch of mathematics, but for problem-solving behavior. Logo is meant to appeal particularly to younger students than APL does, although Logo has also been used successfully with college physics students at MIT.

From the point of view of the "pure" computer scientist, Logo is LISP. The developers of Logo, in fact, have been artificial-intelligence researchers for whom LISP is second nature. The differences between the two languages are all based on the specific intent to make Logo particularly useful as a learning language. Logo's special properties from this point of view will be described next.

Logo is "tuned" for interesting applications. Probably the most famous aspect of Logo is the idea of *turtle geometry*. This approach to computer graphics has been added to other languages, such as Pascal and PILOT, but it originated with Logo.

Most approaches to computer graphics are based on Cartesian coordinates (the "x,y" system you learned for graphing equations in high school—see figure 3). In this approach, each line you want to draw is specified in terms of the specific positions of the endpoints, relative to a fixed-coordinate system. Using Cartesian coordinates, it's not too hard to draw an upright square in a known position, but if the square is tilted, its coordinates must be calculated using trigonometry. The power of turtle geometry is that lines are described not in terms of absolute position in a coordinate system, but relative to the position and direction of the turtle, a conceptual animal that moves around the TV screen. In this system, you don't say where the turtle starts or ends, just how far it moves and in what direction:

```
TO SQUARE :LENGTH
REPEAT 4 [FORWARD :LENGTH
          RIGHT 90]
END
```

Other articles in this issue of BYTE explain more about how turtle geometry works (see "A Beginner's Guide to Logo" by Harold Abelson on page 88 and "Introducing Logo to Children" by Cynthia Solomon on page 196). For our purposes, what's important is that the use of this powerful approach makes graphics programming possible for beginners the first time they use the computer.

In the past, computer programming has appealed to only a small number of people because there has been a real lack of problems that are both interesting and easy enough for beginners. Traditional programming courses have been heavy in algebraic problems ("Write a program to solve quadratic equations."). Therefore, they have not attracted people who

Listing 4: *Pig Latin translation via Logo.*

```
TO PIGLATIN :SENT
IF EMPTYP :SENT [OUTPUT []]
OUTPUT SENTENCE (PLWORD FIRST :SENT) (PIGLATIN BUTFIRST :SENT)
END

TO PLWORD :WORD
IF VOWELP FIRST :WORD [OUTPUT WORD :WORD "AY]
OUTPUT PLWORD WORD BUTFIRST :WORD FIRST :WORD
END

TO VOWELP :LETTER
OUTPUT MEMBERP :LETTER [A E I O U Y]
END
```

don't like the traditional mathematics curriculum.

Turtle geometry is not the only special application built into Logo. Another one is language processing. Letters, words, and sentences are a natural hierarchy of Logo objects. (In most programming languages, by contrast, a sentence is not a list of words, but a string of characters. If you want to deal with the words in the sentence, you have to write a complicated program just to look for spaces in the string to divide the words.) As a simple example, listing 4 is a Logo program to translate a sentence into pig Latin. PLWORD is used as a subprocedure to translate a single word based on this rule: if the word starts with a vowel, add AY at the end. If not, move the first letter to the end and try again.

In the program, WORD and SENTENCE are procedures for joining two objects into a larger object; FIRST and BUTFIRST separate an object into its component parts. The primitive procedure FIRST, when applied to a sentence, produces the first word of the sentence. When applied to a word, it produces the first letter. No other programming language deals so neatly with this hierarchy of objects in human language.

Logo is user-friendly. A language for learners has to be designed to deal with problems that are less important in a language meant for experienced programmers. For example, when you make a mistake, you should get a detailed, helpful error message. Languages that say things like SYNTAX ERROR or ERROR NUMBER 259 are not encouraging to a beginner. Logo has messages like:

+ DOESN'T LIKE HELLO AS INPUT

This means that you tried to add a nonnumber, the word HELLO, to something. When you see the message

I DON'T KNOW HOW TO FRIST

you have used a procedure, FRIST, that you haven't defined. The message

NOT ENOUGH INPUTS TO MAKE

means that the procedure MAKE needs two inputs, and you gave only one. If the error happens during the execution of a procedure, Logo also prints the name of the procedure and the line containing the error.

Since the beginning of time (in 1954), programming students have been getting confused about common programming statements such as X=X+1, a frequently used assignment construct that seems to go against one's algebraic intuition. Pascal's use of := instead of the unadorned equal sign is somewhat of an improvement, and APL's ← is even better. Even so, the notation doesn't make it obvious that X ←3 has an effect very different from X+3 or X−3, which look very similar. In Logo, the assignment is done this way: MAKE "X :X + 1. Although less terse than a single-character symbol for assignment, the word MAKE con-

I-Queue™

...the intelligent buffer that frees your IBM Personal Computer from the time-consuming printing function.

I-Queue accepts data at processor speed (over 4000 characters per second) and sends it to the printer* at whatever speed the printer can accept.

I-Queue can store 16K of RAM, which is approximately 7 pages of average printout (optionally expandable to 65K for storage of 28 average pages). Standard features include complete pagination and formatting capabilities, a reprint buffer, a clock/calendar and many other design enhancements usually found only in sophisticated word processors.

Available direct or from participating PC dealers. Distributor and dealer inquiries invited. For new brochure, write, or call our 800 number.

*Fully compatible with IBM printer and all Centronics interface printers.

IBM Personal Computer is a trademark of IBM.

AVAILABLE FROM STOCK–
WITH CLOCK/CALENDAR &
COMPLETE DEMO PROGRAMS: $495
65K MEMORY EXPANSION OPTION: $175

PERIPHEX inc.
149 Palmer Road
Southbury, CT 06488
800-221-0732 In NY: (212) 961-2103

jures up much more vividly the notion that something is being changed, not just used in a calculation.

There are many more ways in which Logo makes explicit things that many languages leave hidden. For example, Logo uses the colon (which Logoites call "dots") to mean "I want the value of this variable"; the same word without the dots names a procedure. In LISP, the notorious parentheses make it possible to distinguish procedure calls from variable references without the dots notation; most other procedural languages simply prohibit using the same word for both purposes. (That solution would be awkward in Logo because some words like WORD are not only popular variable names, but also names of primitive procedures.)

In any case, according to the design philosophy of Logo, the dots notation is a good thing, apart from its technical necessity, because it calls attention to the fact that a variable's value is different from its name; it also points out that a variable is different from a procedure. For example, in the X=X+1 situation, the two identical-looking appearances of X have different meanings. The second represents the old *value* of X, whereas the first merely *names* the variable being given a new value. In the Logo version, these two meanings are distinguished by the notation. The first is called "X; the second is called :X.

Another example of a distinction that is explicit in Logo and not in some other languages is the division of procedures into commands and operations. An operation is a procedure that computes some value that becomes the output of the procedure. For example, the arithmetic operations are in this category. A command does not have an output, but instead has an effect: it prints something, moves the turtle, or changes the value of a variable.

The same distinction is made in Pascal, in which operations are called functions and commands are called procedures. FORTRAN calls them functions and subroutines. LISP, APL, and C, however, are less fussy. C treats all procedures as operations, but allows an operation to be used as if it were a command; the result of the operation is ignored in that case. In LISP and APL, the result of such a "top-level" operation is printed. (In LISP, every procedure has an output and every top-level command prints something. In APL, some procedures don't have output and, therefore, don't print anything.) In Logo, using an operation without a command is considered an error; if you want something printed, you must use the PRINT command.

The use of infix arithmetic in Logo is a concession to the habits of the users. All other Logo procedures are used in prefix form, with the procedure name before the inputs. Arithmetic can also be expressed in prefix form. The two Logo expressions 3 + 2 and SUM 3 2 are equivalent.

The infix form seems more natural to people accustomed to doing arithmetic outside of the Logo environment. The prefix form, however, is better in some ways. For example, it eliminates the need for precedence of operations (i.e., where division is always done before addition, etc.). Also, it eliminates the need for parentheses to indicate grouping. In LISP, only the prefix forms are used.

Another user-friendly aspect of Logo is its facility for interactive definition of procedures. Early versions of Logo used a line-numbering technique: within each procedure, lines were numbered and could be replaced much as the lines of a BASIC program can be replaced. Current implementations of Logo use a display editor in which special control characters are used to move the cursor around the display screen to change individual characters anywhere in a procedure definition.

Logo has no threshold and no ceiling. This means that Logo is easy enough for anyone to use, but it is powerful enough for any project; it's not a "toy" language. Logo is best known as a language for elementary school children, but it's designed for learners of any age and any level of sophistication.

How young can a Logo learner be?

Well, very young children might have trouble with typewriter keys and with the spelling of procedure names. Several years ago, however, Radia Perlman at MIT built a series of special keyboards with large buttons labeled with pictures instead of words. With this special hardware, she taught the ideas of turtle geometry to 4-year-olds. This project even included the idea of procedures, with buttons called "start remembering" and "stop remembering" to delimit a procedure definition, and one called "do it" to execute the procedure. Multiple procedures could be named by using buttons in different colors.

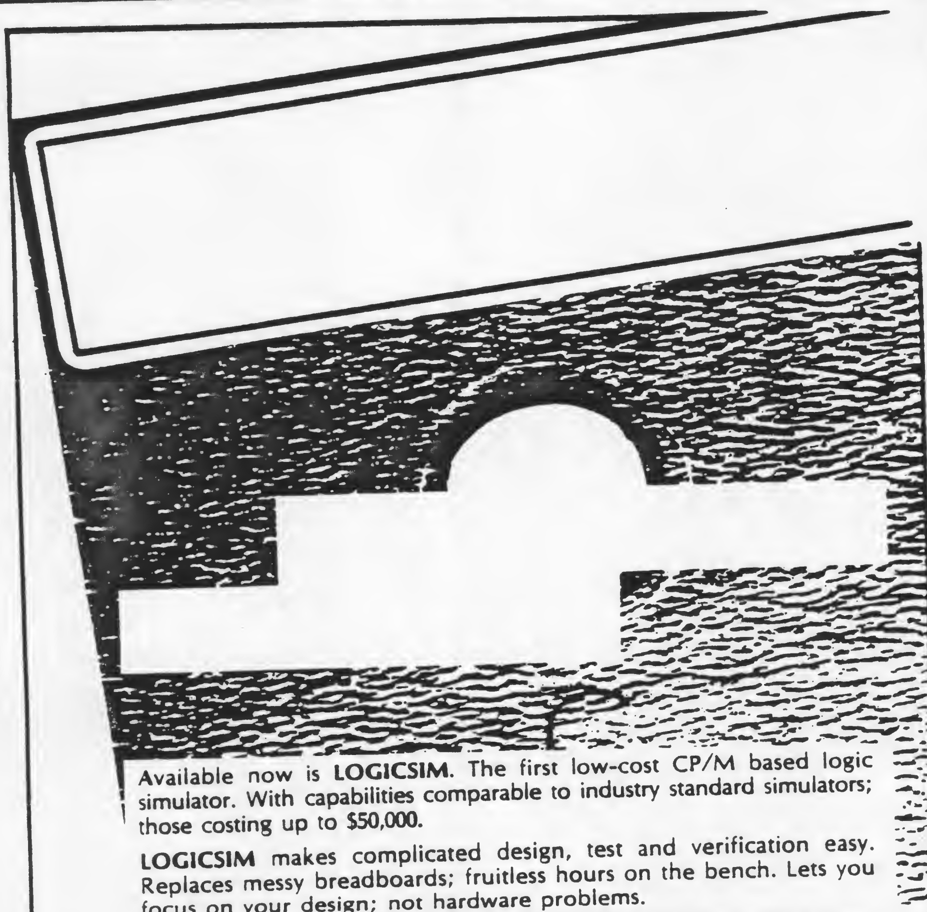
How old can a Logo learner be? Professors Harold Abelson and Andrea diSessa have been using Logo to teach physics to MIT undergraduates. They use Logo simulation programs to demonstrate not only simple Newtonian mechanics but even the general theory of relativity. Their book, *Turtle Geometry: The Computer as a Medium for Exploring Mathematics* (Cambridge, MA: MIT Press, 1981), demonstrates their approach, which has also been used successfully with high school students.

Logo has also been used for a special group of learners, those with severe handicaps. In the past, many children of normal or superior intelligence, but with impaired ability to communicate, have been diagnosed as retarded. Computers can be used with such children both as a communication prosthesis and as a field of interest in which the handicapped learner can exhibit autonomy in pursuing goals. The use of Logo in education for the handicapped is explored in Dr. E. Paul Goldenberg's book *Special Technology for Special Children* (Baltimore: University Park Press, 1979).

Other languages designed with students in mind are BASIC, Pascal, and APL. (I omit PILOT, which was designed not so much for students as for teachers; in its original design, students were supposed to use computer-aided-instruction programs written in PILOT, rather than PILOT itself.) How do these languages compare with Logo in their applicability to education?

BASIC was designed as a modification of FORTRAN for beginners. By far the most important advance in BASIC was its interactive approach. This was much more of a pioneering step than it now seems because people are now accustomed to inexpensive personal computers with this feature. In the early days of BASIC, the only computers were huge, expensive ones. Although timesharing, which allowed several people to use the big computer at once, had recently been invented, many people objected to it because it used the precious time of the huge computers inefficiently. (The response of timesharing advocates was that it was more efficient in the use of human time.) An interactive language was even more time-consuming than timeshared use of the old, compiled languages. For John Kemeny and his colleagues at Dartmouth to move against the general worship of efficiency was very brave.

Besides adding interaction, BASIC removed some of the most difficult parts of FORTRAN. For example, the INPUT and PRINT statements in BASIC don't require a detailed specification of the format in which infor-



Available now is **LOGICSIM**. The first low-cost CP/M based logic simulator. With capabilities comparable to industry standard simulators; those costing up to $50,000.

LOGICSIM makes complicated design, test and verification easy. Replaces messy breadboards; fruitless hours on the bench. Lets you focus on your design; not hardware problems.

Nine logic states, ability to simulate TTL/NMOS/CMOS/PMOS logic, three-state, wired-or and transfer gates with propagation delays allows you to verify design, perform spike and race analysis. Macro circuit capability and expandable macro network library plus numerous other features make it a powerful and usable design tool.

Try it now. Send us a check for $30 and we will mail you a demo copy* of **LOGICSIM** and a User's Manual.

*Please specify 8" IBM 3740/5" APPLE CP/M (13/16 SECTOR)

Introductory price $276. CP/M based 56k to 64k systems (APPLE's with CP/M card).

For more information, call (408) 578-8096. Or write:
E/Z ASSOCIATES, 5589 Starcrest Dr., San Jose, CA 95123

CP/M, APPLE and LOGICSIM are trademarks of Digital Research Inc., Apple Computer Inc. and E/Z ASSOCIATES, respectively.

INTELLIGENT PRINTER INTERFACE

Free Your Computer from the Mundane Task of Printing

Imagine being able to use your computer seconds after beginning an extensive printout.

Visualize your printout with page breaks, page numbering and titles, margins of your choice, indented carryover lines, on any size paper!

Appreciate the time and money you will save by not waiting for your printer.

SooperSpooler, a buffered printer interface, maintains control over your printer while you go on using your computer for more productive activities. Eliminate waiting while your printer pecks through a long document. SooperSpooler accepts information from your computer at up to 3000 characters per second and feeds it to your printer as fast as it can handle it—without using any of your computer's memory or time!

SooperSpooler features include:

- 16K Memory (62K optional)
- Buffer Status Readout
- Space Compression
- Pagination
- Single Sheets
- Headers and Page Numbering
- Indentation on Carryover Lines
- Self Test Routine
- Features also Software Controllable
- Plugs into Most Computer Systems

★ 16K Parallel I/O Unit—$349.00!
★ Serial I/O Option—$95.00
★ 46K Memory Option—$159.00

**SooperSpooler™ by Compulink—
The missing link that gives your microcomputer mainframe printing.**

COMPULINK
CORPORATION

1840 Industrial Circle
Longmont, CO 80501 (303) 651-2014

**Send for brochure
Dealer inquiries welcome
Call for information: 800-525-6705**

mation is presented, as FORTRAN does with its FORMAT statement. (As an example, FORTRAN requires the user to specify the number of digits before and after the decimal point in the printed form of a number.) Of the modern languages, only C uses primarily format-directed input and output. Unfortunately, the important ideas of procedures and local variables were also left out of BASIC.

This means that easy problems are very easy to solve in BASIC, but hard problems are close to impossible. Any large BASIC program is bound to be an unreadable maze of GOTOs. The designers of BASIC, after all, intended it as a language for beginners (i.e., Beginner's All-purpose Symbolic Instruction Code). FORTRAN was supposed to be used for more difficult programs.

The advent of personal computers has pushed BASIC into a more extended role, not because it's easy for the programmer, but because it's easy for the computer! The Logo interpreter, like the Pascal compiler, barely fits in an Apple II computer with 64K bytes of memory. BASIC interpreters are used with 8K-byte machines at a much lower cost. The result is that computer magazines are filled with long, complicated BASIC programs that are far from basic in their readability.

Pascal, on the other hand, was designed to include the most advanced ideas of computer science in recent years. Although intended as a first language, it was meant primarily for college students, particularly those interested in computer science as a career. That helps to explain why it is compiled and typed, two strong barriers to the unsophisticated student. Even the simplest Pascal program is rather complicated to write, enter into the computer, and run. That's why, in practice, Pascal is often taught to students who have already used BASIC and FORTRAN.

BASIC and Pascal were both designed to teach computer programming per se. APL was designed to teach mathematics, especially at the high school level. Its inventor, Kenneth Iverson, used it for several years as a blackboard language without any intention of actually implementing it on a computer. That helps explain his willingness to use special symbols not then found on any actual computer printer. Anything can be drawn on the blackboard!

In its intended use, APL is very powerful. Many computations that require iterative loops and auxiliary variables in other languages can be done in one step in APL. Most people see this power mainly as a matter of terseness; APL is famous (or notorious) for its one-line programs. The real virtue of APL's approach is that it allows the student's attention to be focused on the mathematics of a problem, rather than on the needs of the computer. APL was designed to be used not in a special programming course or a special unit stuck into another math course, but casually throughout an algebra course, just as you'd use a calculator.

Logo's goal is different from all these. It isn't supposed to be an easy introduction to something else, it's not specifically for computer-science majors, and it isn't a tool for teaching the same math curriculum people are already teaching. Instead, it's a door into the territory of the computer as an object for intellectual exploration. To return to the theme stated at the beginning of this article, Logo is for learning learning.

Why Logo?

In his book *Mindstorms: Children, Computers, & Powerful Ideas* (New York: Basic Books, 1980), Seymour Papert says, "It is not true to say that the image of a child's relationship with a computer I shall develop here goes far beyond what is common in today's schools. My image does not go beyond: It goes in the opposite direction." Logo isn't just a programming language; it's also a philosophy of education. Papert's book is the best explanation of that philosophy, but what follows is a briefer summary.

A child learns partly by picking up specific facts and skills. Much of existing formal education is about facts and skills: reading, spelling, and the multiplication table. But a more profound kind of learning is the skill of learning itself, which involves the building of mental models of the world, of oneself, and of the learning process. These models are developed through intellectual exploration. That exploration may begin in a weak, haphazard way, but a good learner develops strategies for purposeful exploration. The more one learns, the better the model of learning, and the more able one becomes as a learner.

In this process of growth, it doesn't really matter what particular aspect of the world you explore. In the introduction to *Mindstorms*, Papert mentions that at age 2 he fell in love with automobile gearboxes. When I was in junior high school, I fell in love with hypnotism. The point about using computers in education is not that everyone must know something about computers, but simply that for many people, computer programming can be the arena for this general process of learning to learn. Because the computer is such a general-purpose machine, it can appeal to many different interests. It can draw pictures, make music, write stories, or move robots.

"I want a job as a computer programmer. Why should I learn Logo, and not something useful like COBOL?" This is a common ques-

to it. The first is that Logo, as explained earlier, is designed to make explicit many of the fundamental ideas of computer programming. Someone who learns Logo is likely to have a very clear idea of the nature of variables, procedures, and most other programming constructs. So Logo may be a better basis even for learning COBOL than simply starting with COBOL itself. But the second answer is that Logo's purpose isn't to train computer programmers. Logo isn't meant to replace all other programming languages.

Logo is generally associated with

This person used to develop applications for the Z-80 market with the best of them.

SOFTECH
MICROSYSTEMS

d.

Circle 373 on inquiry card.

children because most people have a model of the learning process in which children learn and adults don't. This model is unfortunate. Logo can be useful to people of any age, but it will be most useful to you if you approach it in a playful, exploratory way.

It's important to distinguish between the Logo language and any particular implementation of Logo. Some things can't be done in the Apple and Texas Instruments versions of Logo simply because the machines aren't big or fast enough or because the implementation doesn't include some capabilities. For example, no microcomputer version of Logo has a good way of storing data on disk, although all versions can store procedures on disk.

The Logo interpreter barely fits in a 64K-byte Apple II, and the implementation favors the features needed for education, not those needed for practical data processing. But in principle, Logo is a good language in which to develop any application because of its interactive debugging and its procedural style.

Do you want to write a video-game program? It'll probably run too slowly in Apple Logo, unless it's a simple one. But it might be worthwhile to [illegible] with different ideas for your game in an environment that permits quick, easy modification of your program, and then rewrite it later in some other language. The advantage of Logo can be described partly in purely technical terms like "interactive." Another way of looking at it, however, is that Logo encourages the playfulness you need to design the best possible game. If all you want to do is make an exact copy of Asteroids, the benefits of Logo are less important.

In summary: Logo is a LISP-like language, and a laboratory for loose, lifelong learning about learning. ■

This person used to develop applications for the 8086/8088 market with the best of them.

SOFTECH MICROSYSTEMS

Circle 374 on inquiry card.

Introducing Logo to Children

Teaching Logo requires an awareness of different learning styles.

Cynthia Solomon
80 Ellery St.
Cambridge, MA 02138

As computers continue to enter schools and homes, parents, teachers, and children face the problem of inte-ating the machines into their lives. or many, computers serve as powerful instruments for personal use and intellectual development. Many Logo researchers see the potential of computers to serve as personal instruments for everyone and have been working toward that goal. In the process, they have focused on developing not only the Logo language, but things to do with the language and ways of thinking and talking about these activities. How people talk about what they are doing, the way they interact with one another, and the way they interact with the computer give rise to a new kind of culture, a computer culture.

Seymour Papert has been the guiding influence in the development of this kind of computer culture. (See *Mindstorms: Children, Computers, & Powerful Ideas* [New York: Basic Books, 1980] for a fuller discussion of Logo and computer cultures.) Papert created the Logo language for children. Although it had to be simple to learn, it needed a rich and easily expandable vocabulary. It had to reflect some of the important ideas in computer science, such as proceduri-zation, local and global variables, naming, self-referential programming, etc. These are attributes that a language such as BASIC does not possess. BASIC has a reputation for being easy to learn; it has a small vocabulary of key words. But this initial set of key words is not easily expandable; the programmer cannot create new key words. This sets BASIC apart and makes it *easy to learn but hard to use*. The programmer cannot build procedures, name them, and then use them to build other procedures. The powerful problem-solving strategy of breaking problems into smaller and smaller parts can only be a paper-and-pencil strategy in a BASIC programming environment. The structure of BASIC does not support this important problem-solving strategy.

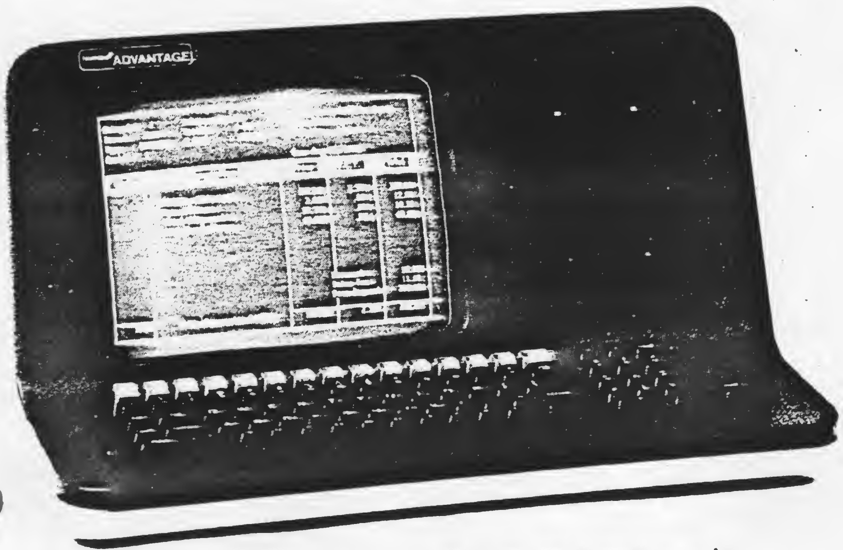
For many, computers serve as powerful instruments for personal use and intellectual development.

Once Logo was developed for children, Papert and his collaborators looked to the computer to provide an environment in which a person could learn by doing and thinking about what they did. A person would actively explore the capabilities of both Logo and the computer by constructing objects and debugging them. The computer would serve as a source or tool for creating interesting mathematical objects that would draw upon a person's intuitive knowledge and that could be used in constructing other objects. One of these objects is the computer-controllable geometric entity, now widely known as a turtle. Exploring the turtle's behavior leads people to draw upon their intuitive geometric knowledge. This knowledge does have a formal aspect as expressed in the area of mathematics known as computational geometry. (See *Turtle Geometry: The Computer as a Medium for Exploring Mathematics* by Harold Abelson and Andrea diSessa [Cambridge, MA: MIT Press, 1981] for an excellent presentation of

bout the Author

Cynthia Solomon, formerly vice-president of Logo Computer Systems Inc., participated in the development of Apple Logo. She is currently finishing work toward her doctorate in education from Harvard University.

NSS Offers An Incredible Advantage.



NSS software, written specifically for Northstar's ADVANTAGE™, outperforms any software available for Northstar computer systems. Our fully integrated accounting package includes general ledger, accounts receivable, accounts payable, inventory control and payroll— **the first Northstar payroll package.**

Why NSS software? If you're a Northstar dealer, it means greater ease in closing sales of Northstar's computer systems within vertical markets. Or, if you own a Northstar computer, it means greater flexibility and expanded capability for your Northstar as a business tool.

Our vertical market applications, such as those we have developed for tire dealers, are the best on the market. Let the ADVANTAGE™ talk to you with NSS' unique cassette training tapes, demonstration disks and sales kits. NSS also provides dealer seminars and a toll free support hotline.

That's The Incredible Advantage!

Order your Introductory Dealer Kit NOW or CALL TOLL FREE FOR MORE DETAILS

1-800-722-3446

NATIONAL SOFTWARE SYSTEMS
P.O. Box 510911
Salt Lake City, Utah 84151

the formal aspects of turtle geometry.) In this kind of a computer culture, people draw upon knowledge acquired in other activities; and they apply what they learn in the Logo culture to many different areas.

My personal contributions toward this goal focus on what is required to encourage the development of computer cultures in the Logo spirit. One of the questions I have considered is: What does a teacher of Logo have to know? There is no one answer to this question. I have seen children teach other children about Logo. Although their knowledge is quite different from the adults who taught them, the children were very successful at sharing with each other a way of thinking and talking about computers. Perhaps I should pose a different question and ask: What are some of the things I think about as I teach people to program?

I see much of my own development as a teacher as acquiring (1) a collection of programming projects that make the power of programming techniques and concepts apparent to beginners; (2) a vocabulary for talking about programming; (3) an awareness of different learning styles and strategies for building on them; and (4) a sensitivity to the kinds of resistances that keep many adults and children from experimenting with mathematical ideas.

A Model for Introducing People to Computers

When I enter a new teaching situation, I have in mind several models of how to introduce people to Logo. I also maintain a willingness to switch from one model to another or even diverge from all of them. My primary goal is for people to do something they could not have done without a computer, but something that is familiar to them (e.g., draw a square or print their name all over the screen). I also want to think toward a next step and how the beginning programmer can build on what happened in the first session in the following one. Flexibility is one of the most powerful ideas in this culture, but to be flexible implies having a model to

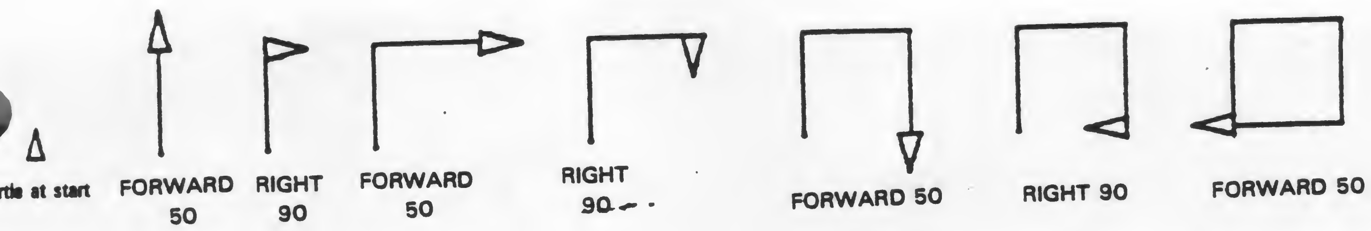


Figure 1: *Building a square. With the turtle at the starting point, this sequence of commands will produce a square. These commands can also be turned into a Logo procedure.*

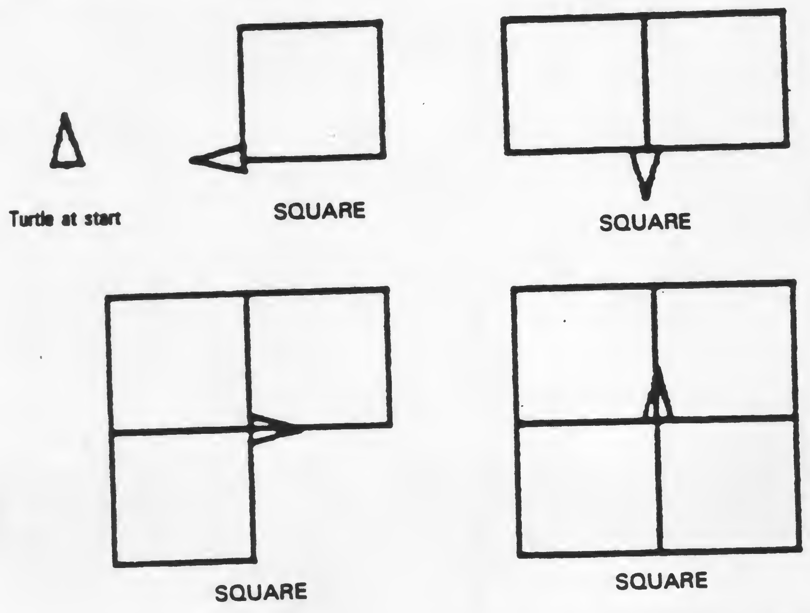


Figure 2: PANES *as evolved from* SQUARE. *The figure is made by repeating the* SQUARE *procedure four times.*

Example of a First Session

Usually, a beginner starts by communicating with a turtle. The student specifies an algorithm in Logo that causes the turtle to draw a geometric design (e.g., a square). This is done relying on intuitive mathematics instead of formula-driven mathematics. That is, the description to the computer is based on how the student would trace out the path of a square if the student had the same limited understanding as the turtle does (e.g., knowing how to move FORWARD or BACK, and turn LEFT or RIGHT). The sequence in figure 1 illustrates the effect of such commands on the turtle: These commands can be named and turned into a procedure that then becomes a part of Logo's working vocabulary:

```
TO SQUARE
FORWARD 50
RIGHT 90
FORWARD 50
RIGHT 90
FORWARD 50
RIGHT 90
FORWARD 50
END
```

The TO informs Logo that a procedure is being created. SQUARE is the name of the procedure. END marks the end of the text of the procedure. Now the programmer might use the SQUARE procedure to produce a new design like the one in figure 2, created by PANES, which follows:

```
TO PANES
SQUARE
SQUARE
SQUARE
SQUARE
END
```

depart from. Thus, I have a model in mind of paths a beginner might take in a first session.

In a first session, I try to convey the following ideas: (1) programming is a process of engaging the computer in conversation using the vocabulary the computer understands; (2) the computer's understanding can be easily expanded; (3) giving meanings to words involves describing a procedure to the computer and giving the procedure a unique name; (4) since procedures created by the programmer can be used like any of Logo's words, they can be incorporated into other procedures; (5) making procedures entails a process of debugging; and (6) pretending to be the computer or the turtle helps in designing and debugging programs.

Thus, in a first session, a person might try out turtle commands like FORWARD and RIGHT. The student would then make a design (create a procedure), give it a name, enter that name into Logo's active vocabulary, and use that new procedure in the construction of others. The teacher, at the same time, is also learning from this experience by thinking about the effect of different actions on the turtle and, with concrete examples and advice, helping the new Logo programmer arrive at a greater understanding of Logo. The teacher also has a chance to rethink the examples presented to the student and develop new approaches and a deeper understanding of the learning process. (See *Apple Logo: Introduction to Programming through Turtle Graphics* by Cynthia Solomon [Pointe-Claire, Quebec: Logo Computer Systems Inc., 1982], which is included in the Apple Logo package.)

The beginner gets an immediate sense of the relationship between program and goal and rapidly elab[illegible]ates this into an understanding of [illegible]e relationships among goals and subgoals, procedures and subprocedures (e.g., the programmer learns that PANES is made by running the SQUARE procedure four times). The beginner is involved in debugging, in learning by doing and thinking about the process. (For example, the programmer may create a square that doesn't close. Listing the procedure, the student catches a bug in the input to FORWARD when drawing the last side of SQUARE; it should be FORWARD 63, not 36.) The programmer may use anthropomorphism or identification as a debugging aid (e.g., pretending to be the turtle causes a person to walk in the same path as the turtle would, and it also helps to understand the turtle's behavior). Thus, in a first session, a programmer writes a procedure that then becomes part of Logo's working vocabulary. In doing this, the programmer achieves a sense of the power available to influence the environment and a sense of accomplishment and creativity.

A Model of Learning Styles

I have observed that children take over the computer in different ways. They show different learning styles, different paths into the computer work. Undoubtedly, this bare statement is true for all learning. What is special here is that the flexibility of the computer allows the process to go

In working with computers, many paths lead to the same goal. Moreover, many equally great goals can be pursued.

further and become more explicit. In working with computers, many paths lead to the same goal. Moreover, many equally great goals can be pursued. This gives children the opportunity to express themselves and explore their own intellectual styles.

Although each child has a unique intellectual personality, and the use of the computer allows us to respect it, we do observe some similarities. I shall describe three distinct learning styles that have emerged, not only from my own work with young children, but from work completed by Dan Watt as part of the MIT-Brookline Logo Project.

The first learning style I call the planner. This child might build structured programs from the top level down or from the bottom level up, but always from a coherent formulated plan.

A second learning style is the macro-explorer. This student likes to mess about with subprocedures or building blocks to arrive at a product.

Whitesmiths, Ltd.

is now shipping Pascal Compilers for 10 (count 'em ten) different operating system families:

ISIS-II*

CP/M, CDOS

Idris/B80

RT-11

RSX-11M, RSTS/E, IAS

Idris/R11, UNIX/V6, UNIX/V7

VERSAdos

Idris/S68K

VMS

UNIX/32V*

*Available in source form only.

Idris is a trademark of Whitesmiths, Ltd. ■ UNIX is a trademark of Bell Laboratories ■ CP/M is a trademark of Digital Research ■ RSX-11M, RSTS/E, RT-11, LSI-11, VAX, and VMS are trademarks of Digital Equipment Corporation ■ VERSAdos is a trademark of Motorola Inc.

All implementations support the full ISO Pascal (Level 0). All pass the Tasmanian Validation Suite with flying colors.

You can partition your Pascal program into separately maintainable files. You can write library functions to add to the extensive set we give you. And you can mix in modules written in other languages, like assembler or C.

In fact, a C Compiler comes with every Pascal Compiler we sell. Prices start at $950.00. Write or call.

Distributors: Australia, Fawnray Pty Ltd. P.O.B 224 Hurtsville NSW 2220 570-6100
Japan, Advance Data Controls Corp. Chiyoda-ku, Tokyo 03263-0383
United Kingdom, Real Time Systems, Newcastle upon Tyne 0632 733131

Whitesmiths, Ltd.

Parkway Towers, 'B', 485 US Route 1 So., Iselin, New Jersey 08830 (201) 750-9000 Telex: 645592

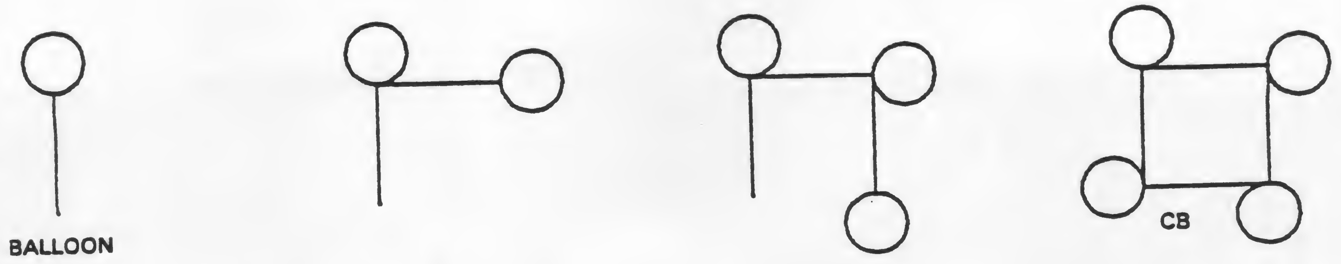


Figure 3: CB *as evolved from* BALLOON. *Using techniques developed to build* SQUARE, *6-year-old Janet designed* CB *from the procedure* BALLOON.

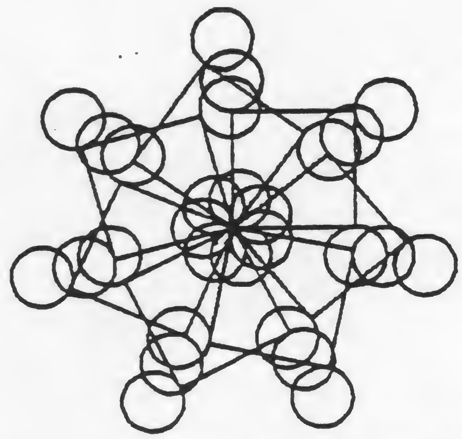


Figure 4: FLOWER, *also designed by Janet, is created by using the procedure* CB, *but turning the turtle 51 degrees and repeating the procedure seven times.*

rather than starting out with a specific goal. In this case, the learner is intent on exploring the effect of the particular building block. Therefore, the result is open-ended.

Finally, some learners have to explore their environment on a micro-level before they can establish patterns of planning or directed exploration. These micro-explorers are often the most timid learners, doing such things as assuring themselves that FORWARD 100 is the same as FORWARD 1, FORWARD 9, FORWARD 11, FORWARD 23, and FORWARD 56. Others might exhibit this conservative, gradual exploration by using the same numbers as inputs to FORWARD and RIGHT, or by repeating the same commands over and over.

Any child might use all three of these learning styles. In an initial session, I might try to "plant seeds" for all three. For example, I would encourage a beginning student to drive the turtle around the screen in a series of direct commands with no goal other than to understand the turtle's behavior. But in the same initial session, I would suggest some concrete goal, such as making the turtle walk in a square or, perhaps, having placed some squares on the screen, I would ask the child to make the turtle touch them. In this, I elicit primarily a micro-explorer style with some hint at a planner style.

I facilitate a macro-explorer style by seizing on something interesting the child has just done and suggesting "teaching" it to the computer. Thus, I encourage the child to form procedures, thereby turning the turtle meanderings into repeatable procedures or building blocks. Using these procedures, the child can create more unanticipated designs.

I would encourage children to follow a planner style of learning by asking them to choose a design from a collection of procedures already familiar to them or by asking them to make a design of their own and use these as procedures. Being sensitive to these styles of learning and their natural intermixing helps to develop strategies for guiding the children. These styles of learning are exhibited by novices to programming in Logo regardless of their age.

Using Procedures

Janet, a 6-year-old, had previously made a square. In constructing CB (see figure 3), she used similar techniques. She made the turtle draw BALLOON and then turn RIGHT 90 repeatedly until it had walked in a complete path and returned to the position and heading it started from. Janet was not aware of this as a generalization, but her specific experiences led her to believe that these two actions repeated over and over resulted in the turtle making a complete trip, i.e., return to its starting state.

Next, FLOWER (see figure 4) was made by running CB, turning the turtle 51 degrees, and repeating these two actions six more times. Why 51 degrees? Well, Janet just happened to pick that number. Why did 51 have that effect on CB? The answer lies in the fact that turning the turtle 51 degrees seven times results in a total turning of 357 degrees, which is very close to a complete rotation of 360 degrees. In this situation, Janet was satisfied; for her purposes, the design was complete.

A more interesting question is: How did she know to probe the turtle environment in this way? She knew certain facts about turtles and turtle-directed procedures that she had gained from her experiences with the turtle. For example, if the turtle draws something and doesn't return to its starting state, repeat the procedure. Something interesting will happen and eventually the turtle will come back to where it was initially. On the other hand, if the turtle does return to its starting state when it makes a design, change the turtle's heading and run the program again. In other words, Janet did not need the teacher's knowledge about the power of 360 degrees. Rather, she needed the idea of the total turtle trip that, translated into intuitive knowledge, told her to keep repeating an action until the turtle returned to its starting state. Janet's learning style in this project

Circle 174 on inquiry card.

DISCOUNT LINE
1-800-528-8960

GUARANTEED LOW PRICES

[illegible]
[illegible] — [illegible] Z80 CARD — [illegible]
VideoCard — [illegible] SmartTerm — [illegible]
[illegible]

ALTOS
8000-2 — $2675 8000-15 — $3850

ANADEX
9500 — $1200 9501 — $1200

[illegible]
[illegible] — [illegible] 800 [illegible] — [illegible]
410 — [illegible] 810 — [illegible]
[illegible] — [illegible] 850 — $162

C-IOTH
F-10P — $1380 F-10S — $1525

DATASOUTH
DS120 — $595 DS180 — $1259

[illegible]
630RO — $1945 630KSR — $2475

[illegible]
Elephant — $20 Scotch — $25 Dysan — $35

EPSON WITH GRAPHICS
MX70 — $289 MX80 — $429
MX80FT — $519 MX100 — $689
GRAPHIC ROM — $69 RS232 — $69

[illegible]
15[illegible] — [illegible] ESPRIT — $595

LOGO
Apple Drive/Card — $350/$50

MODEMS
HAYS — MICROMODEM — $290
HAYS — SMARTMODEM — $229
Pennil — 300/1200 (212A) — $795

MONITORS
Teco-BW — $99 Teco-Green — $115
Sanyo-Green — $249 Color — $425
Amdek-Green — $158 Color — $349

NORTHSTAR
Advantage — $2995 HR64QD — $3025

NEC
8023 — $489 7710 — $2350

MPI
88G — $550 99G — $650

SOROC
120 — $659 135 — $689

TELEVIDEO
912 — $669 950 — $915

TI
810 — $1240 820 — $1795

SOFTWARE
All Major Brands — $CALL

MOUNTAIN HARDWARE
CPS Card — $169 Music System — $429

OKIDATA
M80 — $320 SL125 — $3150
M82A — $450 SL250 — $4200
M83A — $699 M84 — $1059

XEROX
820 — $2399

ZENITH
Z19 — $669 Z89 — $2129

All Prices Subject To Change

EXPOTEK
2723 W. Windrose • Suite 3
Phoenix, Arizona 85029

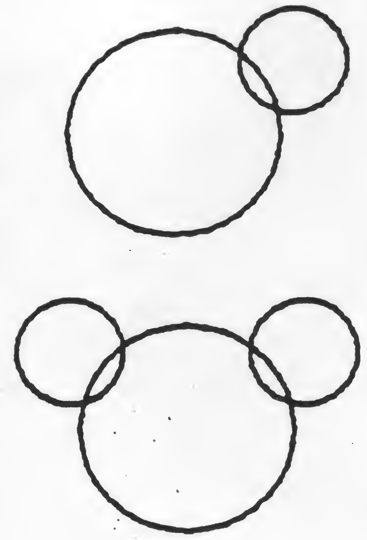


Figure 5: *Initial design for the* BEAR *project. Starting with the design on the top, Lisa added another circle to create an evolving design.*

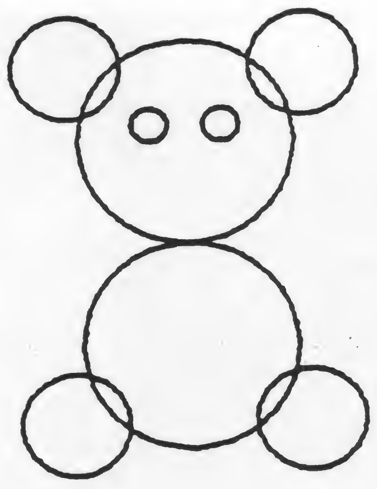


Figure 6: *Completed* BEAR. *Repeating the procedure to create the head, Lisa modified her learning style and created this design.*

and many others was that of a macro-explorer.

The BEAR project was initiated by Lisa, an 11-year-old from an inner-city school close to MIT. Lisa approached the project as a micro-explorer. Although she had previously written several procedures, Lisa showed great resistance to using them as building blocks. For example, after constructing a square procedure, she tried to reconstruct a square, as other micro-explorers might, by telling the turtle to move in incremental steps, e.g., FORWARD 11, FORWARD 9, etc.

I asked her to play with circles using CIRCLER or CIRCLEL procedures, which require the radius as input. I encouraged her to try using circles of different sizes. She made the circles shown at the top of figure 5. I then encouraged her to do the same thing on the other side of the larger circle, as shown at the bottom of figure 5. When asked what it reminded her of, she thought it looked like a bear's head. She then added the eyes and used the head for the body (see figure 6). In so doing, I helped her shift modes from micro-explorer to planner.

This project illustrates clearly that many ways can be used to arrive at a particular goal. Picking a starting state for the turtle influences the construction of the procedure. Whether the job is thought of in terms of subprocedures or whether the design is first created by the student (whether by the mind's eye or on paper) and the turtle is made to trace the path etched on paper has important consequences in how the project is developed.

If the design is to be taught to the turtle by breaking it up into parts, the programmer has to decide what building blocks are needed. This BEAR has several interesting features. It is made entirely of circles. The head and the body are identical. The project is easily changed to focus only on the head or to create, with minor modifications, a different animal (see figure 7).

Both Lisa and I benefited from this project. She became a more confident problem-solver and tended to move away from her micro-explorer style of probing. She used a powerful mathematical idea, symmetry, in a playful but personally meaningful way, and used it throughout her project. I, on the other hand, added to my collection of programming projects. We both followed our intuition. I was

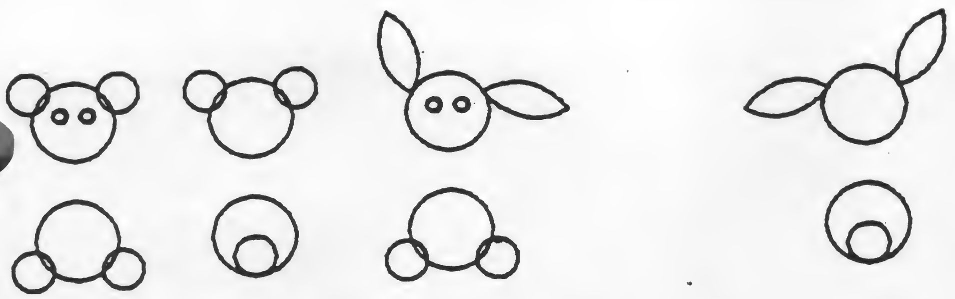


Figure 7: *Other animals created by modifying the* BEAR *procedure.*

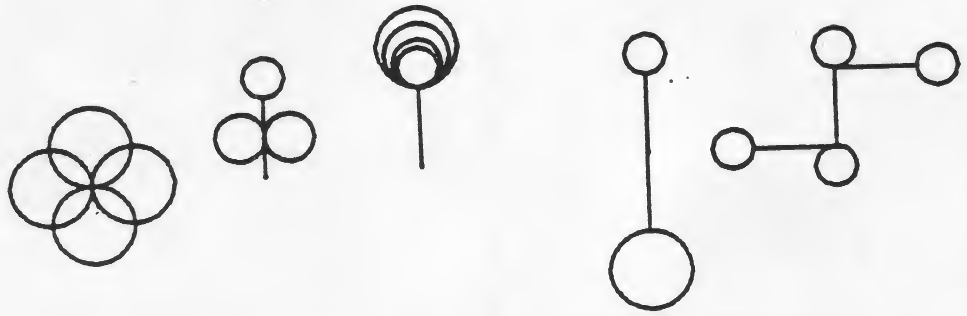


Figure 8: *Other designs created by Lisa using Logo.*

hopeful that when Lisa used circles as building blocks she would discover some lovely thing to make with them, and she did.

A Model of Teaching and Learning as Debugging

Turtle geometry is but one part of the Logo computer culture. Other areas of activity have been explored and many more are waiting to be explored. Turtle geometry, however, serves to illustrate key characteristics of the culture, in particular, the idea of exploring an environment and the objects in it by manipulating them through a complex of interactions based on procedural descriptions. By elaborating the descriptions through debugging (testing procedures in real situations), getting concrete feedback on these actions, and adjusting the initial descriptions to take into account these results, a person's exploration in Logo will be furthered. The process of procedural description and debugging might be seen as a dynamic process of assimilation and accommodation, of making theories and revising them as a result of experience and knowledge, but doing this playfully as an enjoyable activity involving one's whole self.

Conclusions

Sharing in this learning process is a self-empowering experience for all participants. A different way of looking at learning and teaching emerges, one based on the Piagetian idea that even very young children have theories. Thus, teaching and learning are not a matter of being wrong or right, but rather a process of debugging. Learning and teaching are intertwined and become a process of developing debugging aids as knowledge gaps are discovered and filled in. The persons using computers are cast in the role of both student and teacher as they actively participate in development of the computer culture. They contribute to its richness and enrich their own lives.■

The Ultimate Spreadsheet. Never See "Out of Memory" Again.

SuperSoft®

P. O. BOX 1628
CHAMPAIGN, IL 61820
(217) 359-2112 TELEX 270365

First In Software Technology.

Circle 386 on inquiry card.

A Beginner's Guide to Logo

Logo is not just for kids.

Harold Abelson
Laboratory for Computer Science
MIT NE43-805
Cambridge, MA 02139

In the 1960s, computers were very expensive and didn't have much memory. A computer such as the IBM 1620 could store a maximum of 24K bytes (or 60,000 decimal digits). Even the largest research computers could manage only six times that much. Since programs had to use memory sparingly, computer languages were designed to reflect this concern.

Languages had to be simple for the computer, even at the expense of being cumbersome for the programmer. For example, to help the compiler keep track of memory, most programming languages insisted on a close tie between the *names* used in a program and the *storage cells* in the computer memory. As a consequence, the only kinds of data objects that could be directly named and manipulated by program operations were those that could be stored in a single cell. The only data structures available were those whose size could be prespecified at compile time. Most languages also required the programmer to include bookkeeping "declaration" statements, or adhere to other restrictions on the use of names, to make it easy for the compiler to determine what kind of storage each variable required. (For example, some languages required names beginning with I or J to refer to integers, arrays had to be declared together with their size, and defined functions had to have a name beginning with FN, followed by a digit.)

The concern for conserving memory permeated not only the language, but the computer system as a whole. For instance, if the system included a program editor, editing a line of code required the programmer to abort the program, load the editor, read a file, perform the edit, write a new file, exit the editor, recompile the edited code, and reload the program. All this because the editor and the language could not fit into main memory at the same time.

The languages of the 1960s flourished with the personal computers of the 1970s, which, although no longer very expensive, still did not have much memory. As personal computers became more popular, people began to confuse the idea that a language that is simple *for a computer* would also be simple *for people*. ("BASIC has only a few primitives; therefore, it must be easy to learn.") Some people even rationalized that the cumbersome features of such languages were actually advantages. ("Having to declare the data types of variables makes you organize your programs better." "If it's too easy to edit programs, you won't write them carefully in the first place.") And when educators explored the potential uses of computers, they often accepted the drawbacks of these languages as an integral part of programming.

Over the past 12 years, the Logo Group at MIT under the direction of Seymour Papert, along with colleagues at a few universities and research centers around the world, has taken a different approach to educational computing. Rather than accept the limitations of affordable computers (by the standards of those days), we worked with the largest research computers available. The system we used, called Logo, is essentially a dialect of LISP, a powerful language developed for research in artificial intelligence, and used a great deal of memory compared to standards of the 1960s. (Some of the important linguistic aspects of Logo are discussed in "Why Logo?" by Brian Harvey in this issue on page 163. For a more general perspective on LISP, see "An Overview of Lisp" by John Allen in the August 1979 BYTE, page 10.)

In working with Logo, we've discovered some important things. A computer language can be both sim-

About the Author
Harold Abelson, a professor of Computer Science and Education at the Massachusetts Institute of Technology, is also the author of Logo for the Apple II *and* Apple Logo, *introductions to the Logo programming language, published by BYTE/McGraw-Hill Books.*

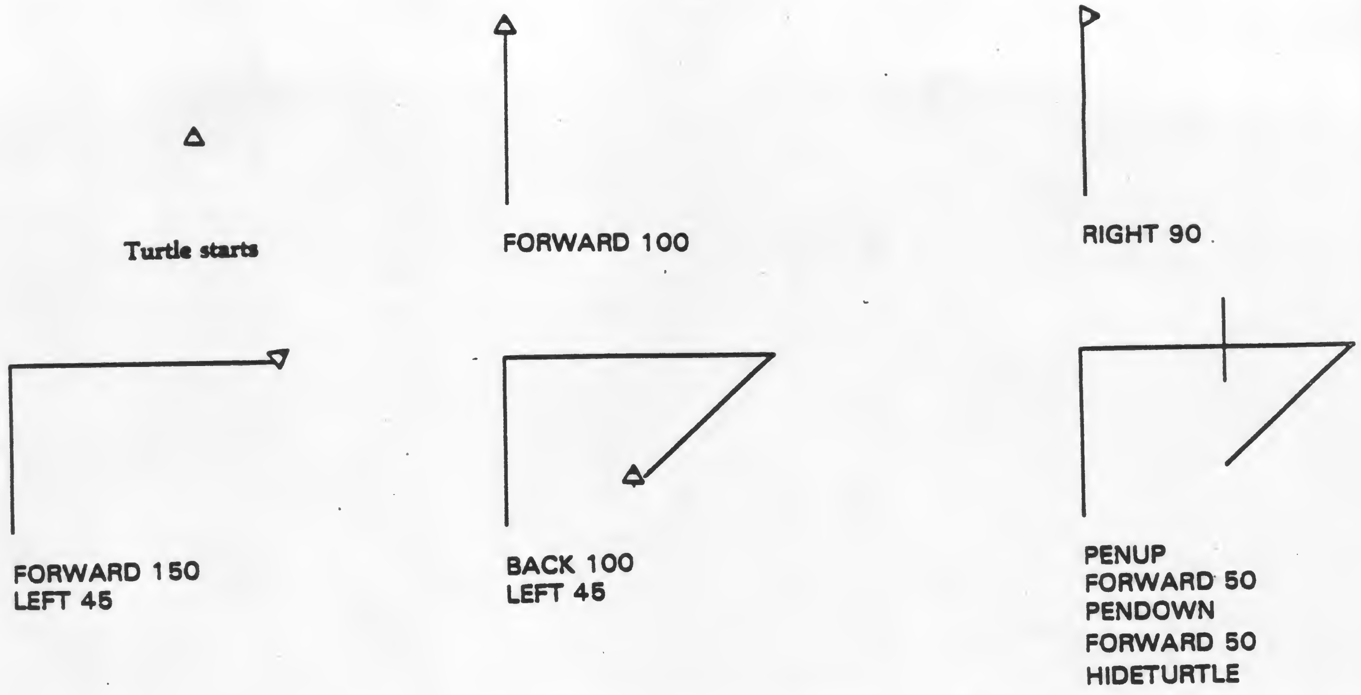


Figure 1: *Moving the turtle with a simple sequence of Logo commands.* FORWARD *moves the turtle in the direction it is facing.* RIGHT *and* LEFT *rotate the turtle.* PENUP *and* PENDOWN *raise and lower the pen—the turtle leaves a trace when it moves with the pen down.*

ple and powerful at the same time. In fact, these two aspects are complementary rather than conflicting because it is the very lack of expressive power in primitive languages such as BASIC that makes it difficult for beginners to write simple programs that do interesting things. More important, we've found that it is possible to give people control over powerful computational resources, which they can use as tools in learning, playing, and exploring. This has often required us to go beyond ordinary considerations of computer-language design to create compelling images of how computation can provide a perspective for reformulating traditional ideas from science and mathematics to make them more accessible, more in tune with intuitive modes of thought. In our research at MIT, working with preschool, elementary, junior high, high school, college students, and with their teachers, we've used Logo to introduce programming and the computational perspective at all levels. In this article, I'd like to show you what it's like to program using Logo, a simple but powerful system, enabling you to explore with a computer.

Drawing with the Turtle

Let's begin with a look at some *turtle graphics*. The turtle is a small triangular pointer on the screen that responds to a few simple commands. FORWARD moves the turtle in the direction it is facing a given number of units. If you type the Logo command FORWARD 50, the turtle will respond by moving forward 50 turtle steps (about 1/4 the height of the screen). RIGHT rotates the turtle clockwise a given number of degrees. BACK and LEFT cause the movements opposite to FORWARD and RIGHT. The turtle also carries a pen, which leaves a trace of its path on the screen as it moves while the pen is down. The commands PENUP and PENDOWN make the turtle raise and lower the pen. Figure 1 shows the result of a simple sequence of Logo commands.

It's lots of fun to make drawings by using these commands (together with a few others, such as CLEARSCREEN, which erases the screen). But in order to really make progress, you have to teach the computer some new words. For instance, you can teach the computer that the turtle can draw a square by repeating this sequence four times: go FORWARD 50 steps, turn RIGHT 90 degrees. The Logo commands would be:

```
TO SQUARE
REPEAT 4 [FORWARD 50 RIGHT 90]
END
```

SQUARE is an example of a Logo *procedure*. The first line (signaled by TO) specifies the name of the procedure. This procedure happens to be called SQUARE (since that's what it draws), but you could have called it anything. The rest of the procedure (the procedure's *body*) specifies the list of instructions to be carried out in response to the command SQUARE; the word END indicates the end of the definition.

Once defined in this way, SQUARE becomes part of the computer's vocabulary. Whenever you give the command SQUARE, the turtle will draw a square.

Procedures with Inputs

An important difference exists between SQUARE and FORWARD. SQUARE always draws a square 50 steps on a side. But FORWARD is more versatile; it takes an *input* that determines how far the turtle should

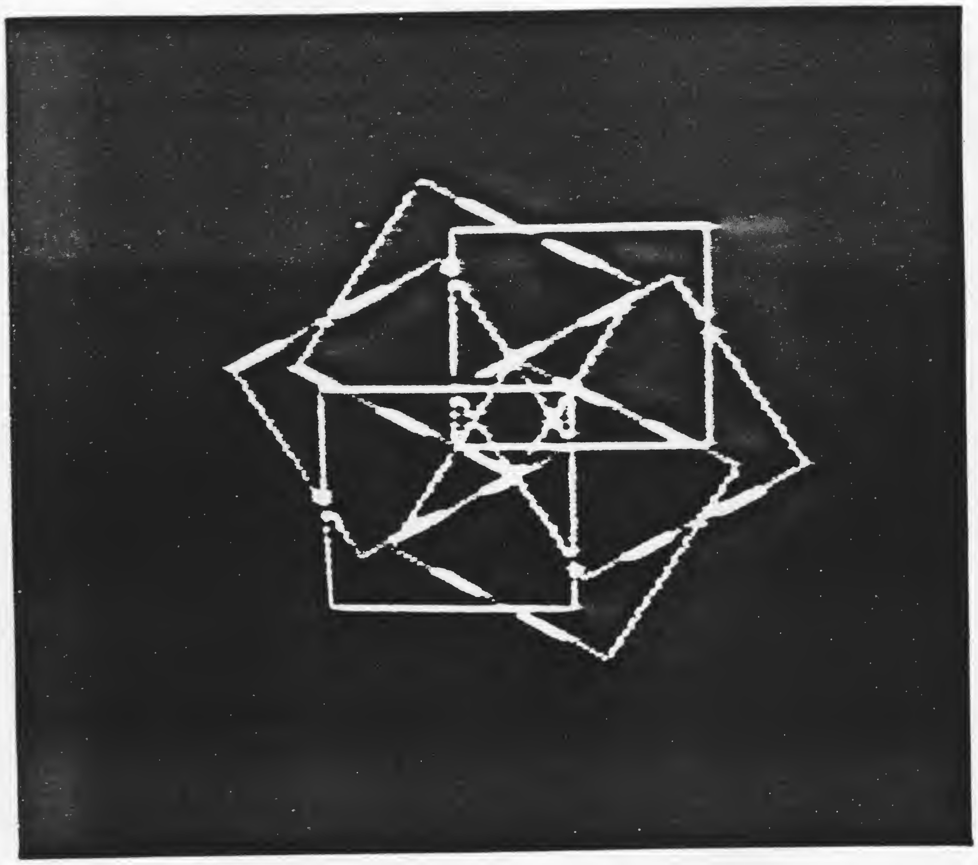


Photo 1: *A design created by a simple one-line Logo program that makes the turtle repeat these steps six times: go* FORWARD 20 *units, turn* RIGHT 60 *degrees, and draw a square of size 75 units.*

move. You can change the SQUARE procedure so that it also takes an input that determines the size of the square to be drawn. For example:

```
TO SQUARE :SIZE
REPEAT 4 [FORWARD :SIZE RIGHT 90]
END
```

You use SQUARE just as you would any Logo command that takes an input. That is, to draw a square with 100-step sides, you type:

```
SQUARE 100
```

To draw a square with 50-step sides, you type:

```
SQUARE 50
```

The definition of SQUARE illustrates the general rule for defining procedures that take inputs. You choose a name for the input and include it in the procedure title line preceded by a colon. Then you use the input name (with the colon) in the procedure body wherever you would normally use the value of the input.

Since a procedure, once defined, becomes just another word the computer "knows," you can use procedures as parts of the definitions of other procedures. Here's a procedure that produces a design by repeatedly going forward, turning, and drawing a square (see photo 1):

```
TO DESIGN
REPEAT 6 [FORWARD 20
  RIGHT 60 SQUARE 75]
END
```

Simple Recursive Procedures

This next procedure also draws a square of a specified size:

```
TO SQ :SIZE
FORWARD :SIZE
RIGHT 90
SQ :SIZE
END
```

Although SQ and SQUARE both draw squares, they behave very differently. Instead of drawing a square and then stopping, SQ makes the turtle retrace the same path over and over, or until you tell the computer to stop. Here is why this happens. When you give the command:

```
SQ 100
```

the turtle must go FORWARD 100, RIGHT 90, and then do SQ 100 again, and so on, and so on.

Add a second input to SQ and you obtain a procedure called POLY, which repeats over and over the sequence: go FORWARD some fixed distance, and turn RIGHT some fixed angle. The procedure takes as inputs the size of each FORWARD step and the amount of each turn:

```
TO POLY :SIZE :ANGLE
FORWARD :SIZE
RIGHT :ANGLE
POLY :SIZE :ANGLE
END
```

To use the POLY procedure, type the word POLY, followed by specific values for the inputs:

```
POLY 60 144
```

Figure 2 shows some of the many different shapes obtained by calling POLY with various inputs.

Recursion is the programming word to describe the ability to use a procedure as part of its own definition. SQ and POLY are recursive procedures of a very simple form—they merely repeat an unchangeable cycle over and over. But recursion is a much more powerful idea and can be used to obtain much more complicated effects. To take just a small step beyond the purely repetitive kind of recursion, consider:

```
TO POLYSPI :SIZE :ANGLE
FORWARD :SIZE
RIGHT :ANGLE
POLYSPI :SIZE + 3 :ANGLE
END
```

Giving the command

```
POLYSPI 1 120
```

leads to this sequence of turtle moves:

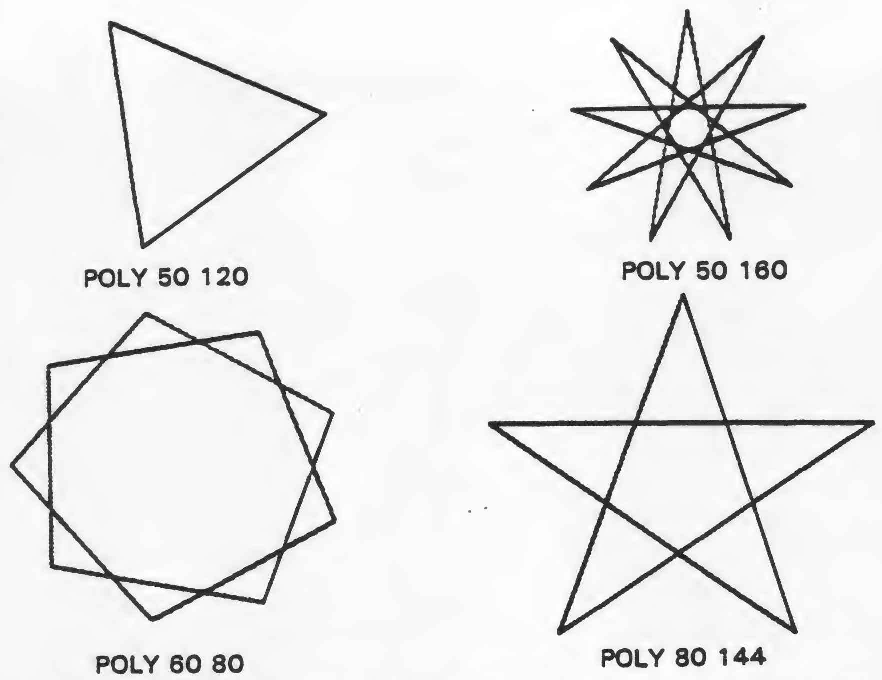


Figure 2: *These shapes are all drawn by the three-line Logo program* POLY, *which has the turtle go* FORWARD *some fixed amount, turn* RIGHT *some fixed angle, and repeat this over and over. The figures drawn by* POLY *always close, but the number of sides that must be drawn before the figure closes depends upon the* ANGLE *input to the procedure.*

```
FORWARD 1
RIGHT 120
FORWARD 4
RIGHT 120
FORWARD 7
RIGHT 120
FORWARD 10
RIGHT 120
.
.
.
```

which produces a triangular spiral in which each of the sides is three steps larger than the previous side. Figure 3 shows some of the shapes generated by the POLYSPI procedure. As a variant, you can replace the FORWARD step in POLYSPI by a command that draws a square:

```
TO SPINSQUARE :SIZE :ANGLE
SQUARE :SIZE
RIGHT :ANGLE
SPINSQUARE :SIZE + 3 :ANGLE
END
```

The result of running

```
SPINSQUARE 1 10
```

as shown in figure 4 is a sequence of squares of increasing size starting with a square of one-step size. Each square is three units larger than the previous one and rotated from it by 10 degrees. The procedure keeps running and the squares keep growing until you tell Logo to stop. You can also modify the procedure so that it stops when the squares become larger than a certain size (e.g., 100 steps) by including a *stop rule*:

```
TO SPINSQUARE :SIZE :ANGLE
IF :SIZE > 100 THEN STOP
SQUARE :SIZE
RIGHT :ANGLE
SPINSQUARE :SIZE + 3 :ANGLE
END
```

Part of the power of recursion is the fact that such simple programs can lead to such varied results.

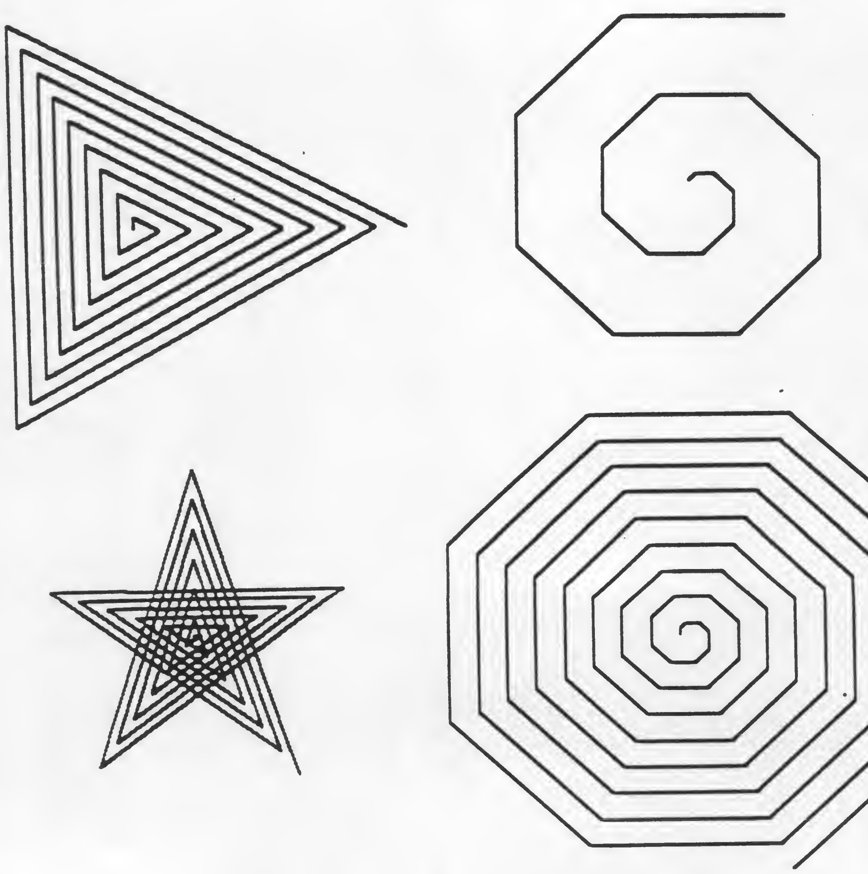


Figure 3: *Figures created using* POLYSPI. *A variant of* POLY *(see figure 2), the program takes advantage of recursion to increase the turtle's* FORWARD *step each time the procedure calls itself. The result is a polygonal spiral. As with* POLY, *varying the* ANGLE *input changes the symmetry of the pattern.*

An Environment for Exploring

As you can see from the examples presented so far, it is very easy to get started programming with turtle graphics. This is partly because of the *subject matter* of turtle graphics. The basic commands have simple, visible effects. At the same time, turtle graphics is an incredibly rich area for

• CRT'S • PRINTERS • TELEPRINTERS •

MICROMAIL'S GOT IT!!

LETTER QUALITY PRINTERS

★ DIABLO ★

• 630 RO
• 40 CPS
• SERIAL RS232C
$1899.00
630 KSR ..... CALL

• 630 API-RO
"INTRODUCTORY OFFER"
$1649.00
"TAILORED TO YOUR SYSTEM"

• 620 RO
"INTRODUCTORY OFFER"
$1149.00
• 25 CPS
• SERIAL RS232C

NEC
• 7700 SERIES
• 3500 SERIES
Call for Our LOW PRICES

QUME
"ON SITE WARRANTY"
• SPRINT 9/45
• SPRINT 9/55
Call for Our LOW PRICES

DEC

LA 34 DA "LA 120 REPLACEMENT"	$899.00
LA 12D .... NEW!	$1599.00
VT100	$1399.00
VT101	CALL
VT131	$1499.00

TELEVIDEO

910	$569.00
912	$689.00
920	$725.00
950	$909.00
925 ..... NEW!	CALL

DEC LA 100
• DUAL MODE
MATRIX PRINTER

★NEW★

3M
WHISPER WRITER
• TWX • TIME SHARE
• COMPUTER TERMINAL

ANADEX

DP 9500/9501	$1125.00
DP 8000AP	$749.00
DP 9620A	NEW!
6000 .... NEW!	CALL

TEXAS INSTRUMENTS

810/2 (includes upper/lower case option)	$1299.00
810/2 VFC/CP (includes u/l case, forms control & compressed print)	$1499.00

"PORTABLES SPECIAL"

T.I. 745	$1399.00
T.I. 743KSR	$849.00
TELETYPE 43 (TTL)	$999.00

VT100 / TELEVIDEO
GPAPHICS OPTION
AS LOW AS
$999.00

CALL TOLL FREE (800) 854-6028

SAVE ON PREPAID ORDERS

MASTERCARD VISA ACCEPTED

C.O.D. REQUIRES 15% DEPOSIT

SHIPPING IS FREIGHT COLLECT

WE SELL INTERNATIONALLY

MICROMAIL

P.O. Box 3297
Santa Ana, CA 92703
Phone: 714/731-4338
TWX: 910 595 1146

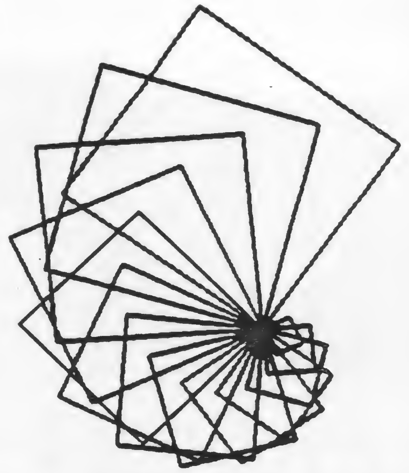


Figure 4: SPINSQUARE *is a simple recursive program that draws a square of a given size, rotates it, increases the size, and continues the process.*

exploration in which even simple programs can have unexpected, often beautiful results. The small amount of Logo we've seen so far is enough to support weeks of activities in programming and mathematics, exploring such questions as "How does the shape of a POLY figure depend on the angle input?" or "Why do so many repeated programs produce symmetric designs?" or simply creating beautiful patterns. Andrea diSessa and I describe some of the mathematics that arises from investigating this computer-based approach to geometry in the book *Turtle Geometry: The Computer as a Medium for Exploring Mathematics* (Cambridge, MA: MIT Press, 1981).

In addition to the subject matter, the *system interaction* also plays a crucial role. When people explore using Logo, they are continually defining new procedures and modifying old ones. A typical compiler-oriented system, in which changing a definition requires switching back and forth among separate editors, compilers, and linking loaders, is inappropriate for this kind of activity. Much of the effort in implementing Logo has gone into providing a programming environment that makes it easy to define and modify procedures. The Texas Instruments and Apple implementations of Logo include integrated screen editors. Giv-

Circle 274 on inquiry card.

Circle 40 on inquiry card.

CP/M OWNERS!

MENU PROGRAMMING LANGUAGE

MPL
ONLY $175

A revolutionary software package that simplifies programming while increasing the versatility of personal computers.

- Runs on CP/M.
- Can be used without any special training in computer programming.
- Displays data menu-style.
- Organizes data by connecting menus together.
- Utilizes menus to drive program operation automatically.

HARD DISK BACKUP

THE BACKFIELD
ONLY $150

A software package that backs up hard disks to floppies without special hardware.

- Runs on CP/M.
- Backs up hard disks to floppies and other disks.
- Can back up all files or selected files.
- Backs up only those sections of the disk that are in use.
- Automatically selects files that have changed since last backup.

AVAILABLE NOW!
(408) 395-0838
OR SEND CHECK TO:

Audio Light, Inc.
146 Town Terrace, Suite 4
Los Gatos, CA 95030

MPL Requires a 24 X 80 CRT.
All software is supplied on 8" single density diskette for CP/M 2.2.
*California residents add 6% for sales tax.

Dealer/OEM Inquiries Welcome.

MPL is a trademark of Audio Light Inc.
CP/M is a trademark of Digital Research

Audio Light

ing the command TO or EDIT activates the screen editor, with the appropriate procedure definition ready to be modified. A single keystroke installs the new definition as a Logo procedure.

The success of turtle geometry is due in large part to the fact that in designing it we did not view ourselves solely as mathematicians and educators attempting to invent a new approach to geometry, nor as computer scientists attempting to implement a system. Instead, we tried to take both perspectives, continually tailoring the computer system to fit the mathematics, and vice versa.

Outputs

We've already seen how to define procedures that require inputs. You can also make a procedure output a value. For instance, the following procedure takes two numbers as inputs and outputs their average:

```
TO AVERAGE :X :Y
OUTPUT (:X + :Y) / 2
END
```

The result returned by AVERAGE can be examined directly (using PRINT) or used in turn as an input for other operations:

```
PRINT (AVERAGE 2 3)
2.5
PRINT (AVERAGE 1 2) + (AVERAGE 3 4)
5.0
PRINT (AVERAGE (AVERAGE 1 2) 3)
2.25
```

Note the Logo convention of using parentheses to group a procedure with its inputs. Although parentheses are almost always *optional* in simple Logo lines, it is a good idea to include them because they make the lines easier to read.

Programming with Procedures

A Logo program is typically structured as a cluster of procedures. These procedures pass information among themselves by means of inputs and outputs. The advantage of this kind of organization is that it separates the program into manageable pieces, as each procedure can be simple in itself. Even in a complex program, it is unusual to have an individual procedure that is more than a few lines long. In addition, the integrated Logo editor and the general interactive nature of the Logo system enable you to define and test individual procedures separately.

To illustrate procedural organization, let's design a simple game that's played as follows. The computer chooses at random a "mystery point" on the screen, and asks the player to make successive LEFT and FORWARD moves with the turtle. Before each move, the computer prints the turtle's distance from the mystery point. The goal is to get the turtle very close to the point in as few moves as possible. Here's a transcript of the game in action. The computer's responses are printed in italics to distinguish them from what the player types:

```
DISTANCE TO POINT IS 67.6
TURN LEFT HOW MUCH?
0
GO FORWARD HOW MUCH?
25
DISTANCE TO POINT IS 90.25
TURN LEFT HOW MUCH?
180
GO FORWARD HOW MUCH?
50
DISTANCE TO POINT IS 47.38
.
.
.
```

And finally:

```
DISTANCE TO POINT IS 12.08
YOU WON IN 11 MOVES!
```

The heart of the program is a procedure called PLAY. This takes as input a number M, which indicates the number of moves so far. PLAY first checks to see if the player has won. If so, it prints a message saying how many moves have occurred, and stops. Otherwise, it asks the player to make a move, and goes on to the next round, with M increased by 1:

```
TO PLAY :M
TEST CHECKWIN?
IFTRUE (PRINT [YOU WON IN]
    :M [MOVES!])
IFTRUE STOP
MAKEMOVE
PLAY :M + 1
END
```

The PLAY procedure is simple in itself

because it delegates the problems of testing for wins and making moves to the procedures CHECKWIN? and MAKEMOVE.

Here's MAKEMOVE, which prompts the user for angles and distances, and moves the turtle correspondingly. It uses a subprocedure READNUMBER, which returns a number typed in at the keyboard:

```
TO MAKEMOVE
PRINT [TURN LEFT HOW MUCH?]
LEFT READNUMBER
PRINT [GO FORWARD HOW MUCH?]
FORWARD READNUMBER
END
```

To check for a win, the program must test whether the turtle's position is close to some predetermined point (e.g., 20 steps). The Logo primitive operations XCOR and YCOR return the turtle's *x* and *y* coordinates. We'll suppose that the *x* and *y* coordinates of the hidden point are given by variables XPT and YPT. If you assume there is a procedure DISTANCE that returns the distance between two points, the CHECKWIN? procedure can be written as follows:

```
TO CHECKWIN?
MAKE "D DISTANCE XCOR YCOR
     :XPT :YPT
(PRINT [DISTANCE TO POINT IS]
     :D)
IF :D < 20 OUTPUT "TRUE
OUTPUT "FALSE
END
```

CHECKWIN? returns as its value either TRUE or FALSE, which is the result that is tested by PLAY to determine whether the game is over. Observe also the use of the MAKE statement to assign values to variables. In this case, D is used to designate the distance.

Here is the procedure for computing the distance between two points, as the square root of the sum of the squares of the coordinate differences:

```
TO DISTANCE :A :B :X :Y
MAKE "DX :A - :X
MAKE "DY :B - :Y
OUTPUT SQRT (:DX*:DX + :DY*:DY)
END
```

Now you need a procedure to start the game:

```
TO GAME
CLEARSCREEN
MAKE "XPT RANDOMCOORD
MAKE "YPT RANDOMCOORD
PLAY 0
END
```

This clears the screen, assigns values (chosen at random) to the mystery-point coordinates XPT and YPT, and calls PLAY with an initial M equal to zero.

The following procedure, used to select random coordinates, returns a random number between −75 and +75. It works by calling the Logo primitive RANDOM to obtain a random number between 0 and 150, and subtracts 75 from the result:

```
TO RANDOMCOORD
OUTPUT (RANDOM 150) - 75
END
```

The only thing needed to complete the program is READNUMBER, which returns a number input from the keyboard:

```
TO READNUMBER
OUTPUT FIRST REQUEST
END
```

READNUMBER uses the Logo primitive REQUEST, which waits for the user to type a line, and then returns a list of all the items in that line. The desired number is extracted as the first item of the input list. (We'll talk about lists below.)

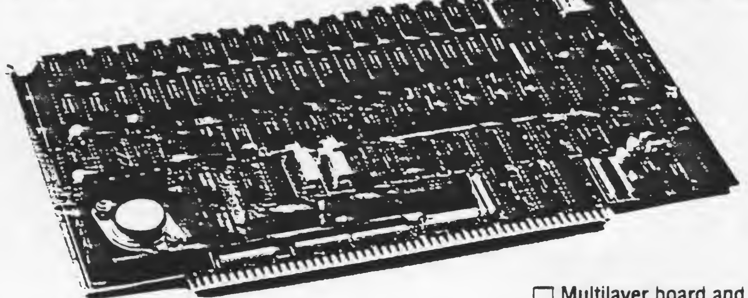
Actually, it might be better to design READNUMBER so that it checks to see if the item to be returned is indeed a number, and to complain otherwise:

```
TO READNUMBER
MAKE "TYPEIN FIRST REQUEST
IF NUMBER? :TYPEIN OUTPUT
     :TYPEIN
PRINT [PLEASE TYPE A NUMBER]
OUTPUT READNUMBER
END
```

Notice the final line of the procedure. Its effect is to make READNUMBER try again for an input until it gets a number, as many times as necessary.

THE MULTI-USER FRIENDLY MEMORY BOARD WITH

NO COMPROMISE



- ☐ 256 Kilobytes
- ☐ 6 MHz with no weight states—Most Processors
- ☐ Designed to operate in *any* S-100 System IEEE or Non-IEEE
- ☐ 16-bit addressing option: Powerful Macrotech Memory Mapping (M^3) bank select architecture allows each 4K block of the 16 bit (64K) logical addresses to be dynamically translated to *any* 4K block of the 256K on board physical memory.
- ☐ 24-bit direct addressing option
- ☐ "Educated" cycle control module generates all timing on board *asynchronously* for maximum access speed and uniform operation
- ☐ Comprehensive technical manual with complete installation guide and source listings for MP/M II™* and "Virtual Disk" solid state disk applications.
- ☐ DMA operation fully supported in strict accordance with IEEE 696 standard.
- ☐ 8/16 data transfer protocol
- ☐ Parity error detection
- ☐ Multilayer board and bus signal filtering for noise-free operation.
- ☐ Full one year warranty

If you wish, send a check or money order for $1,379 (in California add 6% sales tax where applicable).

Manual available at $25.00 each refundable with order.

***Registered Trademark of Digital Research Corp.**
OEM & Dealer Inquiries invited

MACROTECH International Corp.,
22133 Cohasset St., Canoga Park, California 91303
(213) 887-5737

Circle 241 on inquiry card.

From Aspen Software

The Only Complete Document Proofreading System For CP/M®, TRS-80® and IBM-PC Word Processors

PROOFREADER T.M.

Featuring Random House® Dictionary
- Complete Accuracy-looks up every word; does not use less accurate root word analysis, plus Random House Dictionary
- Full Interactive Correction-standard
- Instant Access to Dictionary
- Compact-Random House Dictionary supplied in sizes to fit your system (50,000 words standard)

+

GRAMMATIK T.M.

Beyond Spelling Checking
- Detects typos, punctuation and capitalization errors, misused words and phrases
- Analyzes Writing Style
- Suggests Alternative Usages

= NO ERRORS

"The programs together (Proofreader and Grammatik) offer a dynamic tool for comprehensive editing beyond spelling corrections."
—Dona Z. Meilach in Interface Age, 5/82

"Grammatik is the perfect complement to a spelling check program."
—Dr. Alan R. Miller in Interface Age, 5/82

"If you use a word processor and a spelling checker, then you should investigate the unique capabilities of this program. Grammatik is a surprisingly fast and easy to use tool for analyzing writing style and punctuation."
—Bob Louden in InfoWorld, 12/81

"For the user who is as tight with his dollar as I, Proofreader is the program of choice."
—Stephen Kimmel in Creative Computing, 3/82

"Anyone involved with word processing in any way, whether writing manuals, letters, brochures, newscopy, reports, etc. is encouraged to get the excellent program Grammatik."
—A.A. Wicks in Computronics, 6/82

Grammatik and Proofreader are compatible with all CP/M, MS-DOS (incl. IBM-PC), and TRS-80 word processors. Current CP/M formats: standard 8", Northstar, Omikron, Osborne, Apple. Please call/or write for details of minimum system sizes and availability of additional disk and operating system formats. Shipping costs included. Please specify your system configuration when ordering. Dealers inquiries invited.

	Proofreader	Grammatik	Both
CP/M, MS-DOS	$129.00	$150.00	$250.00
TRS-80 Model II	n/a	$99.00	
TRS-80 Mod. I/III	$89.00	$59.00	$139.00

Random House is a registered trademark of Random House, Inc. Other registered trademarks: CP/M: Digital Research; TRS-80: Tandy Corp.; MS-DOS: Microsoft; IBM: IBM; Proofreader, Grammatik: Aspen Software Co.

Aspen Software Co.
P.O. Box 339-B Tijeras, NM 87059
(505) 281-1634

This makes the game behave as follows:

```
GO FORWARD HOW MUCH?
FJKSL
PLEASE TYPE A NUMBER
FIFTY
PLEASE TYPE A NUMBER
50
.
.
.
```

Lists

We've seen that Logo's procedural organization makes it an easy and convenient language for writing programs. Most modern programming languages are, in fact, procedurally organized, although few languages make it so easy to interactively define and modify procedures as does Logo.

A much more special aspect of Logo is the way it handles *collections of data*. This is done using *lists*. A list is a sequence of data objects. For example:

```
[1 2 BUCKLE MY SHOE]
```

is a list of five things. The items in a list can themselves be lists, as in:

```
[[PETER PAN] WENDY JOHN]
```

which is a list of three items, the first of which is itself a list of two items. Similarly, we can have lists whose items are lists, and so on. Lists, therefore, are a natural way to represent *hierarchical structures*, that is, structures composed of parts that themselves are composed of parts.

Logo includes a number of operations for manipulating lists. FIRST extracts the first item of the list. In this example:

```
FIRST [1 2 BUCKLE MY SHOE]
```

it is 1, and in the next example:

```
FIRST [[PETER PAN] WENDY JOHN]
```

it is [PETER PAN].

The BUTFIRST operation returns the list consisting of all but the first item of the given list, so in

```
BUTFIRST [1 2 BUCKLE MY SHOE]
```

it is [2 BUCKLE MY SHOE], while in

```
BUTFIRST [[PETER PAN]
        WENDY JOHN]
```

it is [WENDY JOHN].

The FPUT operation takes the two objects x and y and constructs a list whose FIRST is x and whose BUTFIRST is y. For example:

```
FPUT 5 [2 BUCKLE MY SHOE]
```

produces the list [5 2 BUCKLE MY SHOE], and

```
FPUT [PETER PAN]
     [BUCKLE MY SHOE]
```

produces the list [[PETER PAN] BUCKLE MY SHOE].

The SENTENCE operation, like FPUT, constructs larger lists from smaller ones, but in a slightly different way. SENTENCE takes a number of lists as inputs and combines all their elements to produce a single list. For example:

```
SENTENCE [PETER PAN]
         [BUCKLE MY SHOE]
```

produces the list [PETER PAN BUCKLE MY SHOE].

The significant thing about lists in Logo is that they can be manipulated as what computer scientists call "first-class data objects." That is to say, Logo lists (as opposed, for example, to arrays in BASIC) can be:

- assigned as the values of variables
- passed as inputs to procedures
- returned as the outputs of procedures

For instance, you can assign names to lists:

```
MAKE "X [OOM PAH]
MAKE "Y [HEIGH HO]
```

and then refer to the values of these variables, so that BUTFIRST :X is the list [PAH]. You can also combine operations on lists to produce more complex operations. For example:

```
FIRST FIRST [[PETER PAN]
        WENDY JOHN]
```

returns the word PETER.

Circle 248 on inquiry card. →

You can also write procedures that manipulate lists:

```
TO DOUBLE :L
OUTPUT SENTENCE :L :L
END

PRINT DOUBLE [OOM PAH]
OOM PAH OOM PAH

PRINT DOUBLE DOUBLE
    [OOM PAH]
OOM PAH OOM PAH
    OOM PAH OOM PAH
```

The implication of this is that you can combine operations on lists, much as you combine operations on numbers in ordinary languages. For example, one very useful list operation is PICKRANDOM, which chooses an item at random from an input list. PICKRANDOM is not provided as a primitive operation, but is easily constructed out of simpler operations, such as finding the length of a list, selecting a random number in a given range, and extracting the *n*th item of a list.

Playing with Text

To illustrate how lists are used, let's examine a program that composes vacation postcards, such as:

```
DEAR DOROTHY
WISH YOU WERE HERE.
LOVE -- JOHN

DEAR MARY
EVERYONE'S FINE.
WRITE SOON -- AUNT EM
```

You begin by setting up lists of names and phrases from which the elements of the postcard will be chosen:

```
MAKE "NAMES
      [JOHN
       DOROTHY
       [AUNT EM]
       OCCUPANT]

MAKE "PHRASES
      [[WISH YOU WERE HERE.]
       [WEATHER'S GREAT!]
       [SURF'S UP.]
       [EVERYONE'S FINE.]]

MAKE "CLOSINGS
      [LOVE
       [SEE YOU SOON]
       [WRITE SOON]]
```

Here's the main postcard program:

```
TO POSTCARD
PRINT SENTENCE [DEAR] NAME
PRINT BODY
PRINT (SENTENCE CLOSING
    [--] NAME)
POSTCARD
END
```

The recursive call in the last line makes the procedure keep printing new postcards over and over. (Compare the SQ and POLY procedures.)

The procedures NAME, BODY, and CLOSING generate the elements of the postcard by selecting items from the appropriate lists:

```
TO NAME
OUTPUT PICKRANDOM :NAMES
END

TO BODY
OUTPUT PICKRANDOM :PHRASES
END

TO CLOSING
OUTPUT PICKRANDOM :CLOSINGS
END
```

You can change the postcard program so that it automatically augments its repertoire of phrases by every so often (say, one chance in three) asking the user to type in a new phrase and adding that to the PHRASES. To do this, add to the POSTCARD procedure the line:

```
IF 1.IN.3 LEARN.NEW.PHRASE
```

The 1.IN.3 procedure returns TRUE with odds of one chance in three and FALSE otherwise. One possible way to write this procedure is:

```
TO 1.IN.3
IF (RANDOM 3) = 0 OUTPUT "TRUE
OUTPUT "FALSE
END
```

Here's how the program learns a new phrase:

```
TO LEARN.NEW.PHRASE
PRINT [PLEASE TYPE IN A NEW PHRASE]
MAKE "PHRASES FPUT REQUEST
  :PHRASES
END
```

The idea is that REQUEST returns (as a list) the phrase that the user types in response to the message. This is added to PHRASES (by means of FPUT), so that the program will be able to use this phrase in future postcards, like:

```
PLEASE TYPE IN A NEW PHRASE
DON'T FORGET TO FEED THE DOG.

DEAR OCCUPANT
DON'T FORGET TO FEED THE DOG.
LOVE -- JOHN
.
.
.
```

Another change you can make is to generate longer postcards, whose BODY consists of one or more phrases. One way to do this is to alter the BODY procedure as follows:

```
TO BODY
IF 1.IN.2 OUTPUT SINGLE.PHRASE
OUTPUT SENTENCE BODY
    SINGLE.PHRASE

TO SINGLE.PHRASE
OUTPUT PICKRANDOM :PHRASES
END
```

This uses recursion in a devious way. Half the time you call BODY, it will output a single phrase, just as before. (The procedure 1.IN.2 is analogous to the 1.IN.3 procedure above.) But the other half of the time, it recursively generates a new BODY and combines this (using SENTENCE) with a single phrase. The new (recursively called) BODY will itself generate a single phrase only half the time. Otherwise, it will call a *third* BODY. The result is that a call to body will generate a single phrase about half the time, two phrases about one-fourth of the time, three phrases about one-eighth of the time, and so on.

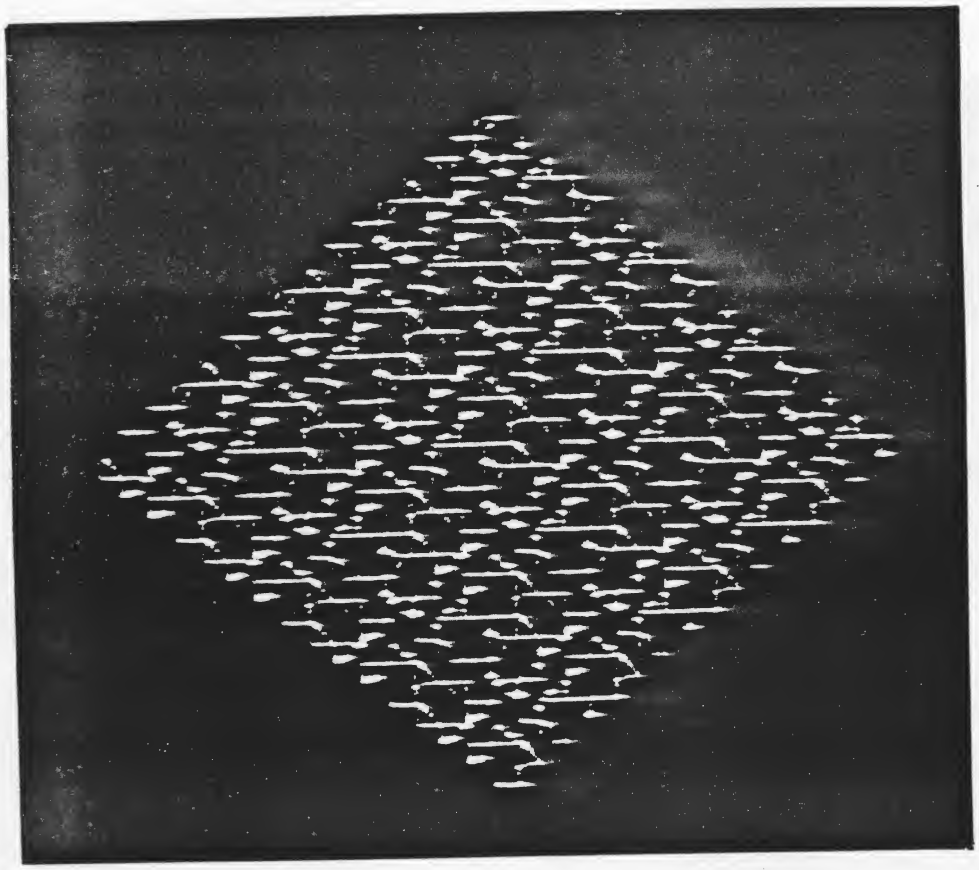
Here's the final postcard program in action:

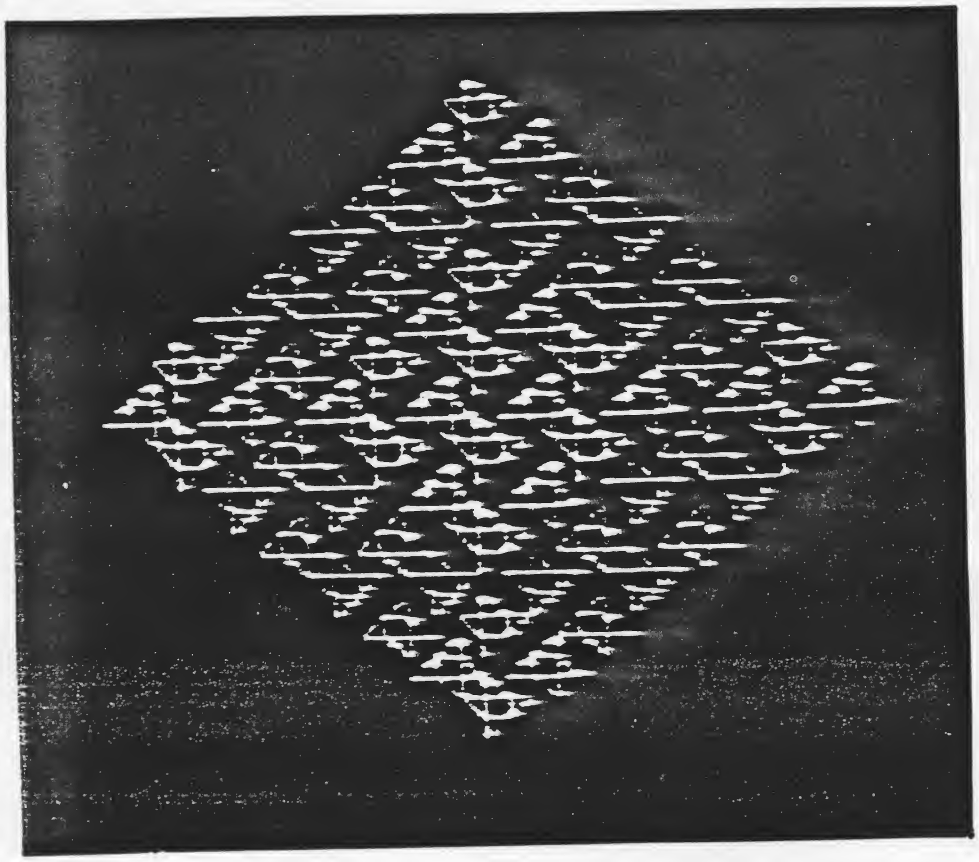
```
DEAR AUNT EM
SURF'S UP. DON'T FORGET TO
    FEED THE DOG.
WRITE SOON -- DOROTHY

PLEASE TYPE IN A NEW PHRASE
GET THE MONEY IN SMALL BILLS.

DEAR OCCUPANT
WEATHER'S GREAT! WISH YOU
    WERE HERE. GET THE
    MONEY IN SMALL BILLS.
SEE YOU SOON -- JOHN
.
.
.
```

Circle 203 on inquiry card.


(2a)


(2b)

Photos 2a and 2b: *Two tiling patterns constructed from the same basic figure (a triangle within a square). The differences in the patterns are due to using different orientations when the basic figure is combined into "higher-level" patterns.*

Text-generation procedures are fun to write and to play with and also easy to modify. You can make them as elaborate or as simple as you like and apply the same ideas to producing essays, poems, love letters, and so on. The idea of generating a random postcard program is based on work done at MIT by Neil Rowe, whose article "Grammar as a Programming Language" (*Creative Computing*, January/February 1978) contains many other examples of text-generation procedures. It also shows how to implement, in Logo, a special-purpose "sublanguage" for creating such programs.

Recursive Tiling

The success of Logo as a catalyst for learning involves much more than the Logo language itself, although the language does play a crucial role. The best kind of Logo activity is a synthesis of programming, mathematics, aesthetics, and, above all, the opportunity to explore. One particularly striking example of this is the "recursive tiling" program invented by Andrea diSessa and Doug Hill. This scheme enables you to write simple procedures that draw patterns such as the ones shown in photos 2 and 3, giving you literally billions of possibilities to examine and explore.

The idea is as follows. Suppose you have a program that draws a pattern inside a square of some given size. By scaling the pattern size in half and gluing together four copies, you obtain a more complex pattern in the a square of the original size, as shown in figure 5a on page 108. In fact, you can generate many different patterns from a single pattern because each copy of the original pattern that you place in each corner square can be rotated through an arbitrary multiple of 90 degrees, as shown in figure 5b.

To convert this idea into a computer program, suppose you have a procedure called PROC that draws a pattern in a square. Assume that PROC takes an input S that specifies the size of the square, scaled so that S is equal to half the diagonal of the square. Assume also that PROC is designed to begin drawing with the turtle at the center of the square pointing at a vertex, and to end with the turtle in the same state.

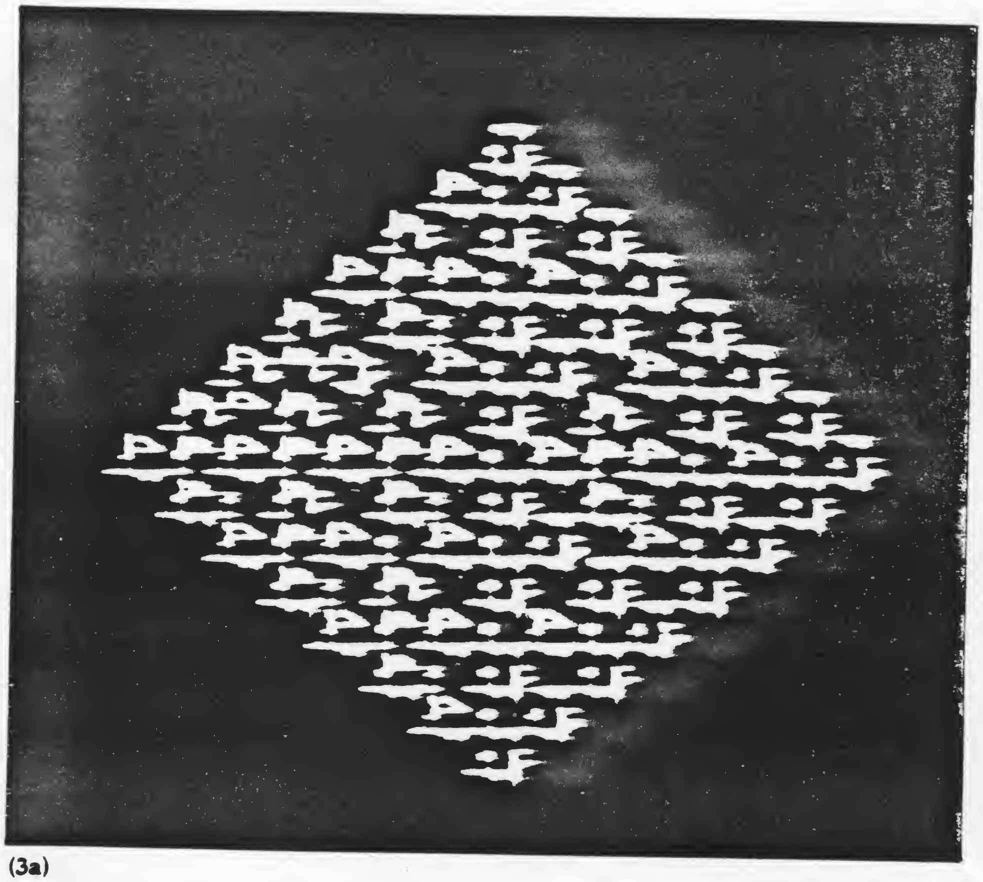
Now suppose you have a square of size S divided into four "corner squares" each of size S/2. The following process is designed to start with the turtle at the center of the square, facing one of the corners. It draws a copy of the PROC design in that corner square and returns the turtle to the center of the larger square. Then it turns the turtle 90 degrees to point at the next corner square. The steps in the process are:

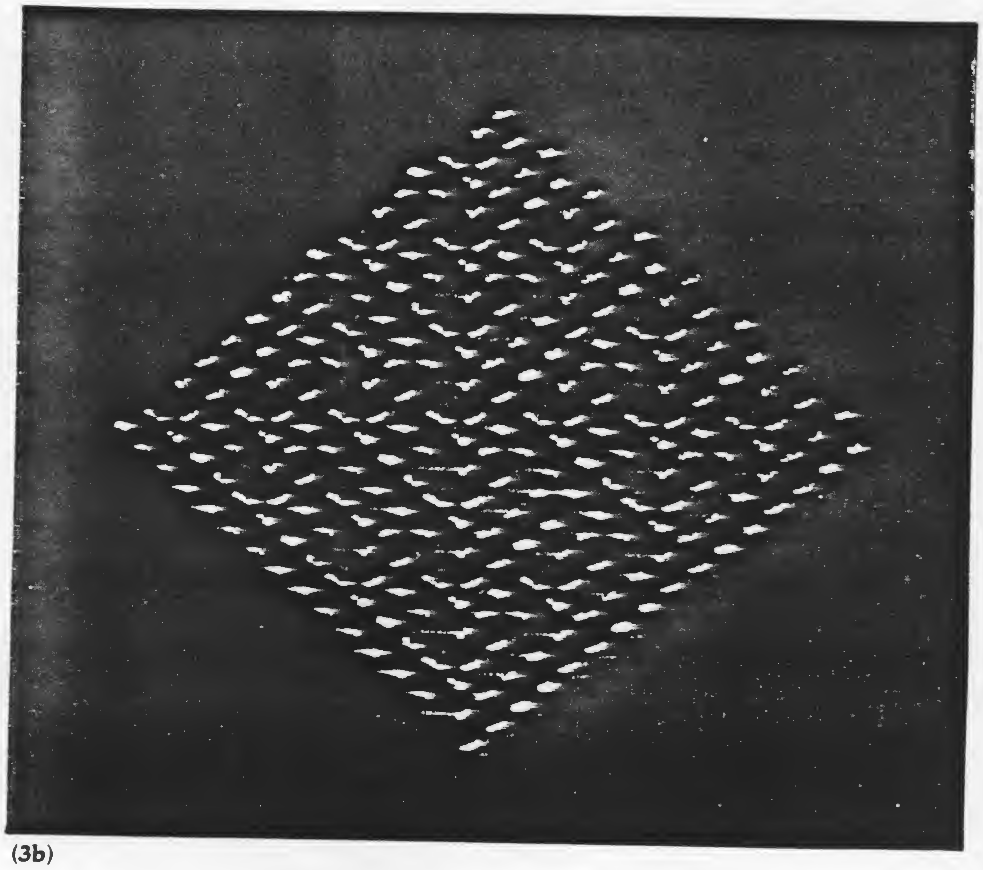
1. Move the turtle FORWARD a distance of S/2. This brings the turtle to the center of the small square, pointing at a vertex of *that* square.
2. Run the procedure PROC with an input of S/2 (half the diagonal of the smaller square). This draws the pattern and leaves the turtle at the center of the small square.
3. Move the turtle BACK a distance of S/2 to return it to the center of the larger square.
4. Rotate the turtle 90 degrees.

In addition, before performing step 2, you can rotate the pattern through a multiple of 90 degrees. If you do this, you should perform the opposite rotation at the end of step 2, so that the turtle will end up facing in the same direction from which it started.

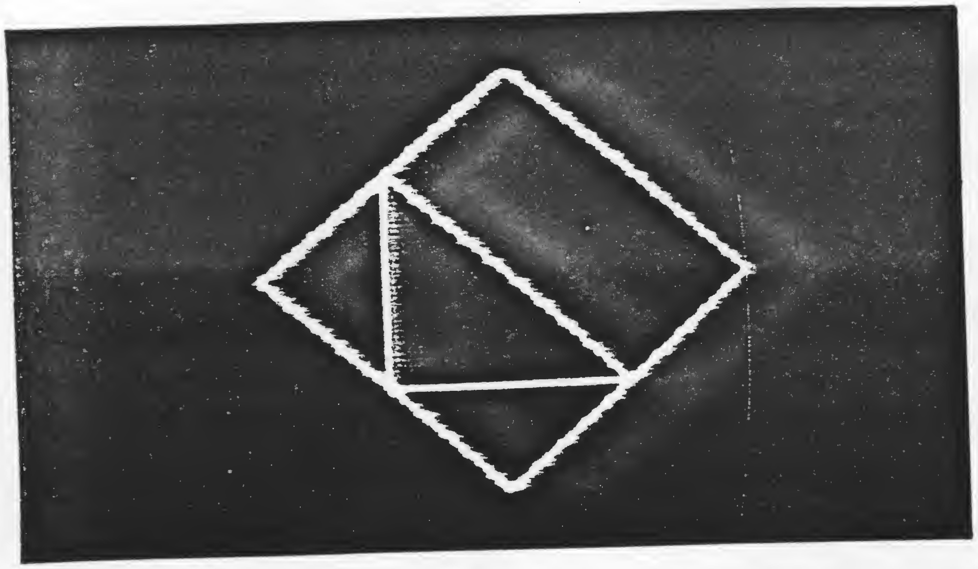
The following CORNER procedure implements this strategy. CORNER takes three inputs. The first, A, is a multiple of 90 degrees to be turned before drawing the pattern (A is an integer from 0 to 3). The next input, PROC, is the name of the procedure that draws the pattern. PROC is assumed to take one input that specifies the size of the pattern. The third input to CORNER is a number S that specifies the size of the square. The procedure is used as follows:

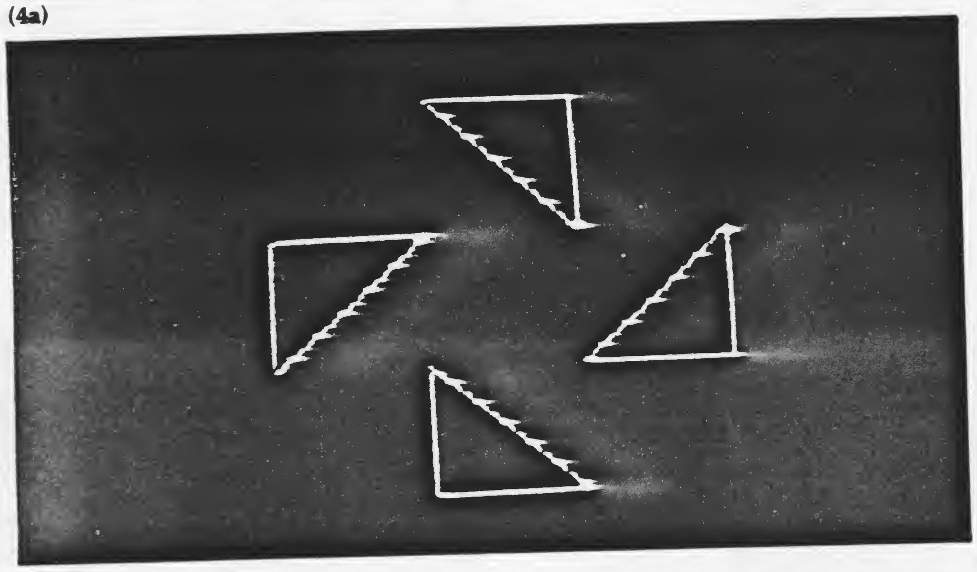
```
TO CORNER :A :PROC :S
FORWARD :S/2
RIGHT 90 * :A
DRAWFIGURE :PROC :S/2
LEFT 90 * :A
BACK :S/2
RIGHT 90
END
```


(3a)


(3b)

Photos 3a and 3b: *Two tiling patterns constructed on the same basic triangle as in photos 2a and 2b. All four patterns use the gluing scheme shown in figures 5a and 5b, and photos 4a, 4b, and 4c, extended to four levels.*


(4a)


(4b)

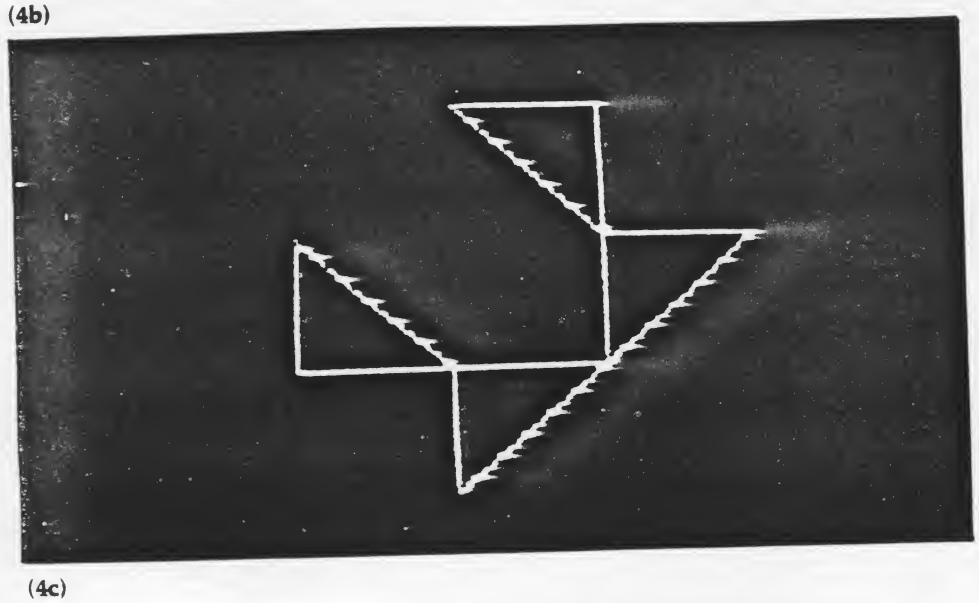

(4c)

Photo 4: *Photo 4a shows a triangle inside a square, as drawn by the procedure* TRI. *(The square, shown here in color, is not drawn by* TRI. *Photos 4b and 4c show two different one-level gluings of the* TRI *procedure to form a more complicated figure inside a square. The complex designs in photos 2a, 2b, 3a, and 3b are four-level gluings based on the same* TRI *figure.*

shown in photo 4a. (In accordance with general gluing strategy, TRI should be a procedure that takes one input that specifies the size of the square.) For comparison, photo 4c shows a different gluing:

```
GLUE2 [TRI] 100
```

Now comes the clever idea. The GLUE procedures enable you to glue together four copies of any pattern. On the other hand, entering

```
GLUE1 [TRI] 100
```

is *also* a command that draws a pattern in a square. In fact, the list [GLUE1 [TRI]], when combined with a size (via the DRAWFIGURE procedure), produces a command that draws a pattern in a square of the specified size. Therefore, you can use a GLUE procedure to glue together four of *these* patterns, for example:

```
GLUE1 [GLUE1 [TRI]] 100
```

or

```
GLUE2 [GLUE1 [TRI]] 100
```

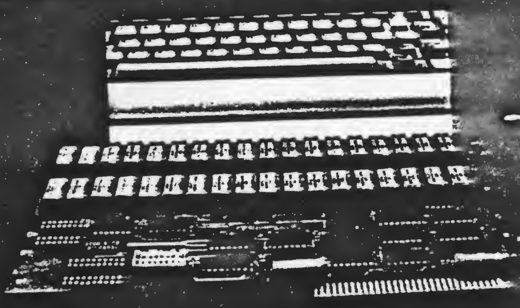
But again, each of these "two-level" gluings is *itself* something that can be glued, so you can make three-level patterns such as

```
GLUE2 [GLUE1 [GLUE2 [TRI]]] 100
```

and so on and so one. The patterns shown in photos 2a, 2b, 3a and 3b are, in fact, all four-level gluings based on the same TRI procedure, using different rotations at the various levels.

There's an enormous range of possibilities to investigate here. Four levels of gluings with 256 orientation choices at each level give 256^4 or more than 4 billion possible four-level gluings, all from a single base pattern! (The number of distinct patterns is reduced by various symmetries in the gluing process, which is itself an interesting phenomenon to explore.) For more variety, you can try different base patterns, or even develop different gluing schemes, such as the one derived from dividing an equilateral triangle into four smaller equilateral triangles. (See *Turtle Geometry*

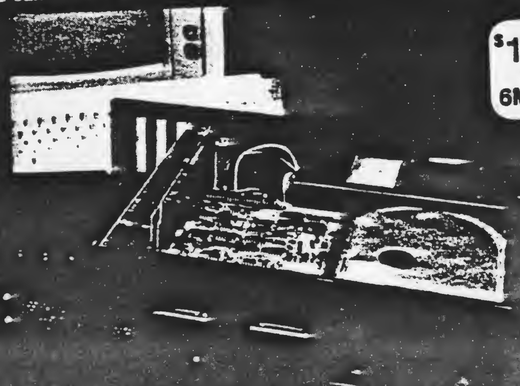
Expand Your IBM P.C.



\$229.00

■ Memory Expansion Board

(PC)²'s Expansion Board offers 4 TIMES the amount of memory offered by IBM P.C. Configurations can be set at 64K, 128K, 192K and 256K Bytes. Board can be easily upgraded in 64K capacities. Upgradability is the key!



\$1895.00 For 6M Bytes

■ Add-In Winchester Disk System

(PC)²'s Add-In Winchester Disk System is housed within the IBM chassis and is easy to install directly into the floppy slot, reducing desktop space. Also available in 12 and 18 M Bytes capacity.

■ Asynchronous Communications Controller

1 or 2 line capability on one card.
Totally compatible with IBM software and diagnostics.

As little as \$85.00

■ DS-Disk Software

(PC)²'s software program doubles the IBM single-sided diskette capacity by using both sides. Increases capacity from 160K to 320K. Used with Tandon TM100-2 disk drive or equivalent.

\$40.00

PHONE OR WRITE—BUT ORDER NOW!

(PC)², P.O. Box 678
510 Lawrence Expressway
Sunnyvale, CA 94086

(408) 749-9313

Postage and Handling included within continental U.S.A.
One year warranty on all board products.
MasterCard, Visa and Checks accepted.

(PC)²

PLUG COMPATIBLES FOR PERSONAL COMPUTERS

Circle 317 on inquiry card.

for more investigations with "recursive designs.")

The Computational Perspective

Logo is often described as a programming language. Those of us who designed Logo tend to think of it rather as a computer-based learning environment, where the activities (exploring the symmetry of POLY) are just as integral as the programming tools used (recursion and lists). Logo is also a continually evolving environment, and the microcomputer implementations of Logo that have appeared during the past year are only the first to be widely available. We plan to extend Logo to incorporate new linguistic features, such as the "message passing" facilities found in Smalltalk and recent implementations of LISP (see the August 1981 issue of BYTE for an overview of Smalltalk), as well as new activities, such as a computer-based physics curriculum that builds upon turtle geometry. At the MIT Laboratory for Computer Science, the Educational Computing Group is designing a follow-on system to Logo suitable for the new generation of personal computers that will be coming into use during the latter half of the 1980s.

The next few years will be exciting ones in educational computing because personal computers are becoming powerful enough to support systems that are designed for the convenience of people rather than for the convenience of compilers. If we can dispel the delusion that learning about computers should be an activity of fiddling with array indexes and worrying about whether X is an integer or a real number, we can begin to focus on programming as a source of *ideas*. For programming is an activity of *describing* things. The descriptions are phrased so that they can be interpreted by a computer, but that is not really so important. Computational descriptions, like those of science or mathematics, provide a perspective, a collection of "tools of thought," such as procedural organization, hierarchical structure, and recursive formulations. Logo, and languages like it, will help make these tools available to everyone. ■

Circle 289 on inquiry card. ⟶

Logo in the Schools

Putting Logo in the classroom has led to some interesting results.

Daniel Watt
Editor, BYTE Books

In the 15 years since its development, the Logo computer language has been used in a variety of research and educational settings. Students from preschool to graduate school; those with severe physical, mental, and emotional handicaps; and students with outstanding ability in science and mathematics have been involved with Logo. It has been used in educational settings to:

- provide an environment for experiential learning of mathematics
- promote the development of problem-solving abilities
- serve as an introductory programming language that helps students learn principles of structured programming
- serve as a vehicle for computer literacy, helping students develop a sense of personal control of a computer
- support the learning of students who, for one reason or another, have not been successful in traditional classrooms
- provide the basis for learning environments in a number of subject areas, including music, language arts, fine arts, physics, biology, and mathematics
- form a foundation for an entirely new kind of school based on Piagetian approaches to teaching and learning, using computers as all-purpose tools to facilitate learning

Each time Logo has been introduced into a school, certain objectives have been emphasized at the expense of others. This article deals with four different Logo projects and describes the settings, the goals of each project, and some of the known results. In some cases, I have drawn on the published reports listed in the references. Where published reports are not available, I have relied on visits and interviews with people directly involved. Each of the projects has had many dimensions that are not included here because of space limitations. For further information, read the reports cited in the references or contact the projects.

Each project had a different type of student population, different choice of goals, and different kinds of results.

The Edinburgh Logo Project, Department of Artificial Intelligence, University of Edinburgh, Edinburgh, Scotland, dealt with 12- and 13-year-old boys attending a private school adjacent to the university. It focused on the use of Logo to create an environment for learning to think mathematically and on developing new methods to teach the content of conventional school mathematics.

The Brookline Logo Project, conducted as a collaboration between the MIT Logo Group and the public

About the Author
Daniel Watt, a former elementary-school teacher, was involved in curriculum development and the Brookline Logo project at MIT. Watt holds a doctorate in engineering from Cornell University and is currently an editor with BYTE.

Photo 1: *Preschool students using TI Logo at the Lamplighter School, Dallas, Texas. Photo courtesy of Texas Instruments Inc.*

schools of Brookline, Massachusetts, had two very different phases. In the first, a laboratory was set up with four computers. The emphasis of the research was to observe and document what a group of sixth-grade students actually learned, rather than assess whether they had achieved a set of preplanned objectives.

The second phase of the Brookline Logo Project involved placing computers in fourth- through eighth-grade classrooms for several weeks at a time. This project emphasized the development of curriculum materials to support the learning of Logo as one activity in a multifocused classroom.

The Computers in Schools Project, conducted by the New York Academy of Sciences in conjunction with the New York Public Schools, provided training for elementary and junior high school teachers to use Logo as a permanent feature of their classrooms. The major focus of this project has been implementation in the school, training and supporting teachers to ensure successful use of Logo in the classroom.

The Lamplighter School Logo Project, conducted at a private school in Dallas, Texas, for students aged 3 to 9, is the most ambitious Logo project to date. Conducted as a joint effort with the school, the MIT Logo Group, and Texas Instruments, the project was intended to provide the school with enough computer hardware that access to computers would not be a limitation on what the students could learn. Logo would be taught to all students and teachers, from nursery school through grade four. Eventually, the project was expected to enhance learning in many areas as it facilitated the use of the computer as a multipurpose learning tool throughout the curriculum. The Lamplighter School also served as the primary test site for the development of the Texas Instruments implementation of Logo.

The Edinburgh Logo Project

The objective of the Edinburgh Logo Project was to discover whether the students' ". . . ability to do mathematics and to talk about their mathematics was changed by exploring mathematical problems through [Logo] programming." The quotes in this section are taken from *Teaching Mathematics Through Logo Programming: An Evaluation Study*, by Howe, O'Shea, and Plane (see references section 1). The students were a group of 11 sixth-grade boys from the George Herriot School, a private school near the university. They were selected from the school's lowest-level math group.

The project lasted for two years, during which the students attended a Logo lab at the university. For the first year, the students worked through a set of graded worksheets to learn the basic elements of Logo. For the second year, they did special Logo exercises designed to teach topics selected from their regular mathematics curriculum.

The project was highly structured in several respects. The students' learning experiences were structured by means of assigned worksheets that they worked through in order, each at his own rate. In this way, researchers could effectively monitor the progress of each student. During the second year of the project, Logo activities were drawn from mathematical topics such as areas of rectangles, factors and multiples, positive and negative numbers, and plotting coordinates on graphs.

The research aspect of the project was also carefully structured. Students were given standardized tests in mathematics before and after the project. Their progress was compared with that of a control group (drawn from boys in the second lowest-level math group). Both groups of boys, as well as their teachers, were also given a series of questionnaires designed to measure their attitudes toward mathematics. Great care was taken to see that the research design was carefully carried out.

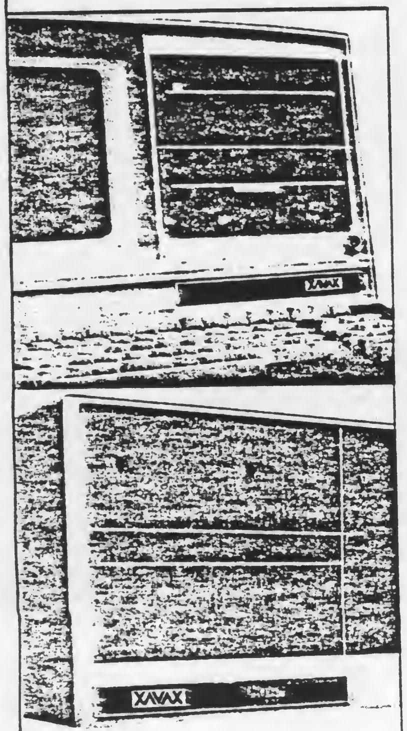
The published results of the project on student achievement were not very dramatic. Over the two years, the experimental group improved a bit more than the control group on a "basic maths" test. The reverse was true on a "maths attainment" test. The most interesting finding had to do with the teachers' perceptions of the students in both groups. Teachers found that students who had taken part in the Logo classes were more willing to "argue sensibly about mathematical issues" and to explain their "mathematical difficulties clearly." This finding may have depended as much on the teaching approach used by the Logo teachers—as compared with the classroom mathematics teachers—and on the individual assistance the Logo students received, as it did on the Logo activities themselves.

Teachers found that students who had taken part in the Logo classes were more willing to "argue sensibly about mathematical issues" and to explain their "mathematical difficulties clearly."

Conversations with some of the people involved indicate that a lot of interesting data about what and how the students learned was collected during this project. Unfortunately, little of that information has been analyzed or published. For people interested in teaching Logo, the most tangible results of the project may be the sets of worksheets developed to teach Logo concepts and mathematical applications. These represent a useful set of Logo teaching ideas—even if they are not used in the strictly

FORTRAN, Pascal, COBOL/Ada™*

UNIX™III 68000



XAVAX Three
Hardware
- 68000 CPU at 8MHz (10MHz option)
- Two level memory management
- 256KB on board RAM (512KB option)
- Up to 32KB of on board PROM
- 64KB RAM, bus expandable to 1MB
- Floppy disk, 6MB Winchester disk
- Full Multibus™ and multimaster
- 9 serial I/O ports
- 9 slot backplane (IEEE 796)
- 84MB Winchester hard disk
- 10MB cartridge tape backup
- 168 and 300MB hard disks

Software
- UNIX™III, multiuser version
- UC Berkeley utilities
- Screen oriented vi editor
- FORTRAN-77 (programs to 1MB)
- ISO Pascal, UCSD extensions
- RM COBOL, C, assembler
- *Ada™ under separate OS
- XCCX™ multiversion file backup
- TXT™ enhanced UNIX™ editor

Multibus™ of Intel Corporation
UNIX™ of Bell Laboratories
Ada™ of Department of Defense
Ethernet™ of Xerox Corporation

GRAPHICS 1024 x 792
- Emulates Tektronix 4014-1
- 1024 x 792 pixel landscape display
- 4096 x 4096 apparent resolution
- Bit mapped raster graphics
- Mouse pointer for display changes
- 17 inch interlaced monitor (P40)
- X,Y (0,0) coordinates displayed
- Full screen update 64 milliseconds
- Dedicated floating point hardware
- Ethernet™ local area networking

IMMEDIATE ATTENTION! B882

☐ XAVAX One (board only, 128KB RAM)	$2,000	☐ XAVAX Three (UNIX™III)	$14,000
☐ XAVAX Two (with 10MB Winchester)	$9,500	☐ XAVAX Four (multi-CPU)	$40,000
☐ TekGraf™ (desktop graphics system)	$12,000	☐ Ethernet™ (local network)	$3,000

☐ Manuals: XAVAX One ($15), XAVAX Two ($35), XAVAX Three ($75) USA postpaid.
Mail checkmarked copy: XAVAX, POB 1105. MasterCard and VISA accepted.

Phone: (319) 323-2332 **XAVAX CORPORATION** 300 Northwest Tower, Bettendorf, IA 52722 TWX 910-525-1177 sequential format for which they were originally designed. Copies of student worksheets used in both years of the project are available from the Edinburgh Logo Group.

The Edinburgh Logo Group has also been involved in several other educational projects. In one project, student teachers who were not math specialists were taught Logo to see how it would affect their teaching of mathematics. In another project presently under way, computers have been installed in several schools so that the Logo curriculum can be taught by classroom teachers who have taken a Logo training course. This project is intended to give clearer results about the impact of Logo on the improvement of classroom performance in mathematics.

In order to carry out the current study, the Edinburgh Logo Group implemented a version of Logo on the Terak computer system, an LSI-11-based system with high-resolution graphics. Disks for this version of Logo are available from the Edinburgh Logo Group. Other Logo implementation projects are under way for microcomputer systems widely available in Great Britain.

The Brookline Logo Project

The first Brookline Logo Project, funded by the National Science Foundation and conducted by the MIT Logo Group in collaboration with a public school in Brookline, Massachusetts, had a very different set of goals and results. In this case, 50 sixth-grade students were given the opportunity to learn Logo in a computer lab established within the school. The work of 16 of these students, representing a full range of academic abilities and interests, was selected for study.

The entire Logo learning experience of these students was carefully monitored and analyzed, documenting what the students learned, what learning styles they used, and what types of choices they made. Some common material and ideas were presented to all the students and introductory turtle geometry projects were stressed at the beginning of the project. Students also had the opportunity to choose their own activities and went on to develop many different Logo projects, including a math quiz, word games and conversations, animations, geometric explorations, tic-tac-toe, and dynamic action games. The students were expected to be in charge. The teachers were there to help them accomplish their own goals.

The results of the project indicate that Logo learning environments are suitable for many different kinds of students. All students, ranging from those who were academically gifted to those who had the poorest academic records, were successful in the Logo classes. The surprising success of students with learning disabilities led to a separate proposal to provide Logo training and equipment for teachers who specialized in this area. The final report of the project summarized the students' learning styles and analyzed what they learned in the areas of computer programming and mathematics. A second volume of the report traced the learning experiences of each of the 16 students individually. The report provides a basis for an introductory Logo curriculum, as well as a rich source of project ideas suitable for students with widely divergent interests and abilities.

The Brookline Logo Project was not very successful in obtaining "objective" data about learning gains made by the students. Standardized tests had been rejected as irrelevant to the goals of the project (the ability to use turtle geometry is not measured by sixth-grade math tests). The problem-solving tests and mathematical tests devised and administered by the project staff had inconclusive results. The problem of developing objective tests in such areas as problem solving or procedural thinking is still an open question for educational researchers.

Another limitation of the project was that it required an extremely sensitive and knowledgeable teacher, with a great deal of time to consider the needs of each student. It was the

1. DISPLAYDEF - Development Utility
- Define screen formats
- Define entries, prompts, messages, text
- Document screen format
- Build screen files

2. DISPLAYASM - Powerful assembly calls
- Build screens, process entries
- Handle messages and prompts
- User callable directives and primitives
- Screen fields transparent to program

- Large system approach
- User friendly
- Language and system independent
- Complete entry checking, conversion & protection
- Low RAM and Disk overhead

- High speed operation
- Screens easily changed & tested
- Professional documentation
- Maintenance free
- Low cost - $175.00 with Manual

- CP/M™ 1.4 or 2.2 with cursor addressable terminal or CRT
- Languages - Microsoft Basic-80 and BASCOM. Other popular languages to follow

MCFILE - a data base definition system with user directives
MCPRINT - a page oriented print system with user directives

BASIC 80 and BASCOM
are products of Microsoft
CP/M is a trademark
of DIGITAL RESEARCH

MasterComputing Inc.
P.O. Box 17442
Greenville, SC 29606
(803) 244-8174

Circle 247 on inquiry card

Vive la Difference.

Micros may all seem alike, with their Z-80 logic, S-100 bus, and [illegible] operating system.

Until you take a hard look.

Only Delta is the "Private Computer." You won't find it in [illegible] department store. We market exclusively through [illegible] who add value to our product.

Delta fields a complete product line, from single-user [illegible] 64-user distributed processing networks.

Delta offers an expanding software library, [illegible] software development assistance, [illegible] and support.

Delta's flexible financial program [illegible] business, keeps your profits up.

Some of our competitors may claim [illegible] difference [illegible]

But Vive la Difference.

DELTA PRODUCTS INC.

[illegible]

hope of the project staff that the two-volume report, with its analysis of student learning and many examples of student projects, could be an effective resource for teachers working in less ideal settings. The report was also intended as the basis of a Logo curriculum to be developed in subsequent projects.

The second Brookline Logo Project was also funded by the National Science Foundation to develop a curriculum supporting classroom use of Logo. Computers were placed in classrooms from grades four through eight. Teachers were provided with a small amount of training, and the project developed curriculum materials to be used by students and teachers. During the project, two computers circulated among several classrooms. Each classroom had exclusive use of a computer for 8 to 12 weeks. During this time, students worked on their own at the computer, individually or in pairs, while the rest of the class went on with its regular work. About once a week, the entire class met for a lesson at which ideas would be shared, new concepts introduced, and assignments given.

The curriculum materials developed by the project are at two different levels: an introductory Logo curriculum for grades four through six, and a set of advanced Logo projects based on playing and modifying a set of "dynaturtle" games. The introductory curriculum includes step-by-step instructions for students, as well as a number of different types of project ideas. Teachers are given information about everything from the physical arrangement of the computer in the classroom to the concepts the students will be learning, suggestions for whole class lessons, and a checklist to help them monitor student progress.

The advanced activities focus on a series of dynaturtle games that can be used in two different ways. The games provide a microworld in which students can explore the behavior of the dynaturtle—a Logo turtle that has been programmed to follow Newton's Laws of Motion. Each game introduces a new factor to be considered. The first game involves making the dynaturtle hit a target, which forces a student to learn to control its momentum and understand something about how the vector quantities of force and momentum are combined. The second game involves driving the dynaturtle around a circular racetrack, introducing some of the concepts involved in orbital motion. The third game, a version of the familiar Lunar Lander, introduces the effect of gravity. The booklet accompanying the games contains many suggestions and challenges for the students that are designed to help them understand the physics concepts embedded in the games. (Also, see R. W. Lawler's "Designing Computer-Based Microworlds," in this issue on page 138.)

Another method for using the dynaturtle games is as a programming project. The games are deliberately designed to be simple so that they lend themselves to many obvious improvements. Every student who has played them has had ideas for making them better and more interesting. A student booklet provides detailed suggestions for making a series of changes in each game. Students who have already learned simple Logo programming can learn some of the intermediate features of the programming language while using these games as models for the construction of elaborate programs from small modules. Students who have gone through these projects are ready to tackle any number of interactive programming projects of their own devising.

Curriculum materials developed during the project are not yet publicly available. The MIT Logo Group is seeking a commercial publisher for them in accordance with the requirements of the National Science Foundation.

One of the most interesting aspects of the second Brookline Logo Project was the way in which students emerged as Logo teachers. Because there was a group of "student experts" at the beginning of the project, seventh graders who had participated in the first project, teachers incorporated these students as tutors into their planning from the start.

As the project went on, certain students from this group (and others) became known as experts at Logo programming and at managing the computer systems. Teachers throughout the school routinely began to ask these students for help when necessary. When the youngest students in the school, the fourth graders, were introduced to Logo, each student was assigned an upper-grade tutor for the first few weeks. Thus, the fourth graders developed a quick proficiency with the mechanics of the system and were able to begin their own projects very quickly.

A related aspect of the project was the way that students in the same classroom worked together on Logo activities. During the first Brookline Project, student interaction had been limited by the arbitrary manner in which groups were assigned to the laboratory. In the classrooms, students formed natural groupings to share ideas and help each other. Project ideas and "secret knowledge" of how to do certain things were passed among the students by word of mouth. The result of using students as teachers and working partners was a reduction in the teachers' role as source and authority, and the creation of a student-based Logo culture.

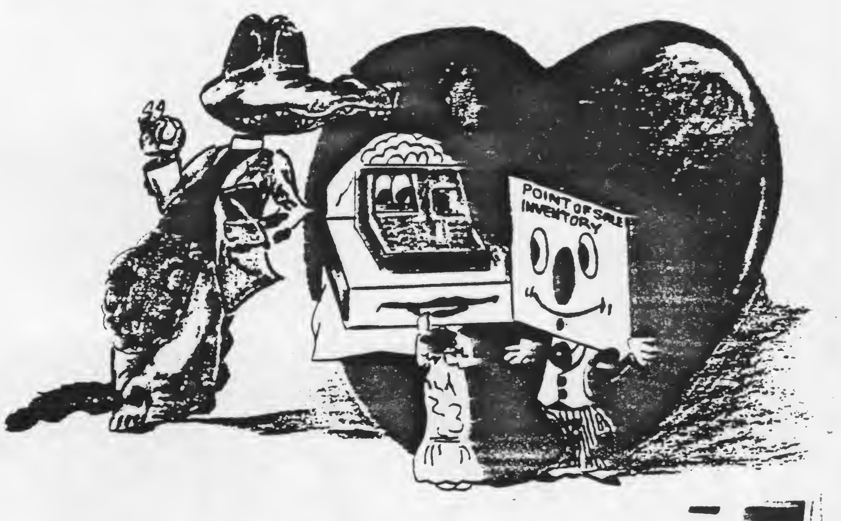
It had been assumed at the start that *teacher* knowledge would be a major limiting factor in what the students could achieve. It turned out that this was not the case. The limitations on *student* knowledge were what limited what other students could learn. A strategy was devised to support the transfer of knowledge from student to student. Once a week, an after-school student interest group met to work on projects and share ideas. This gave the students involved an opportunity to further their own Logo knowledge, to increase their store of project ideas, and to develop more consistent ways of thinking about how Logo works. All this made them much more effective in their informal role as spreaders of the Logo culture.

Because the project focused

"The Perfect Marriage"

ARBA Register and Accounting Plus*.

Great hardware now with superb software. ARBA register, the affordable, dependable RS232 on line cash register. Accounting Plus* Inventory Control software, comprehensive, timely, accurate. The ARBA point of sale software module that brings the two together. The ability to integrate on line real time inventory control with a total business package: G/L, Payables, Receivables, Payroll, Purchase Order, Budgeting, L/Y Comparisons, Electronic Spread Sheet.



ARBA

ARBA Fine Business Computing Corporation
390 E. Roosevelt Road
Lombard, Illinois 60148
(312) 620-8566

Runs under CP/M** or MP/M**

ARBA Register—$1295.00—Suggested Retail

Dealer Pricing Available

* TM Software Dimensions, Inc., Citrus Heights, Ca.
** TM Digital Research, Pacific Grove, Ca.

primarily on curriculum development, there was no opportunity for a study of the role of students as teachers and the impact of this on the roles of classroom teachers as traditional authority figures and sources of all school learning. This type of situation is becoming quite common; students know more about the computers than their teachers because they have more time to develop and share their expertise. The use of students as teachers should be a serious consideration for teachers, researchers, and curriculum developers as computers continue to spread into the schools.

Computers in the Schools—New York City

The Computers in the Schools Project, conducted by the New York Academy of Sciences in collaboration with New York City School Districts 2, 3, and 9, provides teachers with training and support to teach Logo in their own classrooms. The project involves students in grades two through nine from a full range of socio-economic backgrounds. Like the second Brookline Project, the computers are located in elementary and middle school classrooms. A major difference is that the teachers have had an extensive training period and each classroom is assigned a computer for the entire year.

The project began in the summer of 1980 with a three-week training program for 11 teachers and a principal. An expanded training program in the summer of 1981 included eight more teachers from each of the three school districts. During the year, project staff members made weekly visits to each participating classroom. Teachers also attend a monthly seminar held at the New York Academy of Sciences.

Although the project has not yet published any progress reports, the staff believes it has been successful in

---

For The Best In Price, Selection and Delivery,

Call Now TOLL FREE

800-368-3404

(In VA, Call Collect 703-237-8695)

AMPEX•INTERTEC•TEXAS INSTRUMENTS•GENERAL DATA COMM.•ANDERSON JACOBSON•C. ITOH•QUME • BEEHIVE• DATASOUTH•DIABLO•CENTRONICS • NEC • PRENTICE

SUPERBRAIN MICROS

INTERTEC:

64K DD*	ONLY $1975
64K QD*	ONLY $2350
64K SD* (96TPI)	ONLY $2750

*(includes M/Soft BASIC)

DDS-10 Meg (Hard Disk)	ONLY $3195

PRINTERS

NEC:	
7710	$2196
7715	Call for Special Price
7730	$2196
7720	Call for Special Price
7725	Call for Special Price
Std. Forms Tractor	$ 200
3510	$1690
DATASOUTH:	Call
DIABLO: 630-R102	$1995
630-R110	$1795
630-R153*	$1745
*(for IBM P.C., Apple II, TRS-80)	
630-K104 (KSR)	$2385
620-SPI	$1195
QUME:	
Sprint 9, 35 KSR	$1840
Sprint 9, 45RO, Lim. Pan.	$1845
Full Panel	$1969
Sprint 9, 55 F.P.Ex.Mem.	$2186
Sprint 9,55RO,Ltd.Ex.Mem.	$2095
Bi-Dir. Forms Tractor	$ 199

TERMINALS

AMPEX:	
Dialogue 30	$ 775
Dialogue 80	$ 939
BEEHIVE: (SMART DISPLAY)	
DM5	Call
DM5A	Call
DM310 (3101 Emulator)	Call

NOTE: IBM and Burroughs compatible terminals available. Please inquire.

C. ITOH	
CIT 101	$1350
TEXAS INSTRUMENTS:	
745 Standard	$1390
745 Std. (Reconditioned)	Call
810 Basic	$1249
810 Package	$1439
820 Package RO Package	$1610
820 KSR Package	Call
840 RO Basic	$ 795
840 RO Tractor Feed Pkg.	$1059

COUPLERS

PRENTICE STAR: 300 Bd.	$ 124

DISC DRIVES

QUME:	
Data Trak 5	$300 or 2 for $549
Data Trak 8	$525 or 2 for $999

SOFTWARE

BISYNC-3780	$ 769
Wordstar	$ 289
Data Star	$ 199
Mail Merge	$ 99
Spell Guard	$ 229
Plan 80	$ 249
Super Calc	$ 249
Wordstar (IBM P.C.)	$ 289
Mail Merge (IBM P.C.)	$ 99
d Base II	$ 529
CalcStar	$ 189
SuperSort	$ 186
Nevada Cobol	$ 176

Special! While They Last!

SOROC TERMINALS

IQ 120	ONLY $625
IQ 130	ONLY $525
IQ 140	ONLY $989

In addition, we can make EIA RS232 or RS449 cables to your order, and supply you with ribbons, printer stands, print wheels, thimbles for all printers listed. And many, many more items. CALL NOW.

All items shipped freight collect either motor freight or UPS unless otherwise specified. VISA and MasterCards welcomed: add 3% for credit card purchases. Virginia residents, add 4% Sales Tax. For fastest delivery, send certified check, money order or bank-wire transfer. Sorry, no C.O.D. orders. All equipment is in factory cartons with manufacturers' warranty (honored at our depot.) Prices subject to change without notice. Most items in stock.

MasterCard **TERMINALS TERRIFIC** VISA

Terminals Terrific, Incorporated, P.O. Box 216, Merrifield, VA 22116. 800-368-3404 (In VA, Call Collect 703-237-8695).

90 to 95 percent of the classrooms involved. In a conversation with project coordinator Michael Tempel, he defined "success" in the following terms: "The positive educational benefit was obvious! Kids were engaged in valid intellectual and social processes. You could see them developing. . . . We have seen striking changes in kids' relationships to schools and learning; kids who had not been successful in school got turned on."

Like the second Brookline project, the Computers in the Schools Project found that interaction among students has been a major positive consequence of having Logo in classrooms. Although Tempel stressed that to remain effective the Logo environment requires "measured and periodic input from the teacher," he has been struck by how much work the students do without teacher intervention. The activity "has a real quality of self-sufficiency" for the students.

One important condition of the project has been the insistence that each classroom have at least one computer for an entire year. Tempel believes that access to computers has been a major element in the success of the project. Another condition was that all the teachers involved had to volunteer for the project and take the summer training without additional pay. This helped ensure that teachers had a direct personal stake in the project. Such factors should not be underestimated when comparing this to other Logo projects or considering it as a model for implementing Logo in other school districts.

When the formal project ends this year, the teachers who have already been trained are expected to carry out future training and support activities on their own. Teachers in each of the three districts will have the responsibility for training and support in their own district. The Logo Learning Center, established by Logo Computer Systems Inc., will function as an informal meeting place, providing a mechanism for teachers to stay in touch, share ideas, and receive additional training.

The future of the Computers in the Schools Project itself involves an ambitious proposal to create a "magnet school" for the three districts in which the students would have access to computers from the earliest grades. With specially trained volunteer teachers, the school would be a focus for Logo-related research and curriculum development. This project has received the support of the three school districts involved and is presently in the proposal-development stage. Since costs for equipment, research, and curriculum development will be far beyond what can be provided by the school system, the New York Academy of Sciences is seeking support from a number of different groups. It hopes to be able to start with a small number of students this fall.

The Lamplighter School Logo Project

The most ambitious Logo project to date was carried out jointly by the MIT Logo Group, Texas Instruments, and the Lamplighter School, a private school in Dallas, Texas. Lamplighter School has 400 students between the ages of 3 and 9. The school has been provided with 50 Texas Instruments Logo systems that are used throughout the grades. The goal of the project is to establish a setting in which student access to computers would not be a limiting factor and to see what students could learn in such circumstances.

The project is now in its third year. A half-time teacher/coordinator oversees the day-to-day workings of the project and provides individual Logo tutorials for every teacher in the school on a biweekly basis. Computers are in every classroom from the nursery school through fourth grade. Every teacher and child is involved to some extent.

On a recent visit to the school, I was struck by just how comfortable the children are with the computers. Two 4-year-old girls were using a computer to construct geometric designs on a screen with square-shaped sprites. (A sprite is a hardware implementation of a turtle, to allow multiple moving objects on the screen.) Nearby, classmates were engaged in more conventional activities: building with blocks, putting together a puzzle, playing with toy

---

CP/M USERS LOOK OUT: S/S-PASCAL IS FASTER

AND REQUIRES ONLY 32K OF MEMORY

	Execute (seconds)	Total Size (bytes)
S/S-PASCAL	12.5	1117
PL/I 80 *	14.0	5977
PASCAL MT+ *	19.0	3043
PASCAL/Z *	109	3328

Re: BYTE, September 1981, page 180
A high level language Benchmark

* CP/M, PL/I 80 TM of Digital Research
PASCAL MT+ TM of MT Microsystems
PASCAL/Z TM of Ithaca Intersystems

S/S-PASCAL Version 3.0 for 8080/8085/Z80

Write to get the whole story, or better send $30.00 for
— Demo discette with compiler (READ statement not included) and sample programs
— Tutorial manual (over 100 pages)

DON'T TAKE OUR WORD FOR IT, compile your own programs and test them before buying full compiler

S/S-PASCAL Version 3.0: $295 - Demo discette: $30

P. O. Box 197
Armdale, N.S.
Canada B3L 4J9

SCOTIA SOFTWARE

(902) 425-3879

- **Standard PASCAL features *plus***
- Generates optimized native code
- Compiles directly into .com file
- External assembler subroutines
- Random access files
- 14 digit BCD floating point reals
- Chaining and overlays
- Dynamic strings
- Powerful string manipulation
- Direct access to memory
- Address and size functions
- NEW, DISPOSE, MARK, RELEASE
- and more . . .

"We answer all inquiries the same day"

Dealer inquiries welcome.

cars, playing house, and finger painting. Computers for these young students are just another way of exploring their world.

The typing ability of the first and second graders is amazing. The students are already writing simple programs, using the keyboard and the Logo screen editor with great dexterity. One second-grade "hacker" had just invented a procedure that simulated the effect of the reset key, clearing the screen and printing "Welcome to Logo!" He also proudly pointed out, "It doesn't erase your procedures!"

In the third grade, several children were clustered around two computers. One of them had made a "secret" animation program that made a number of sprites move continuously in a dynamically unfolding spiral. Three boys were trying to duplicate the procedure on the adjoining computer. Another child was designing a sprite shape for the center of the screen that would look as if it were emitting the spiraling sprites. Competition, cooperation, communication, problem solving, programming, geometry, and artistry were all happening at once. Meanwhile, the teacher who had introduced the basic idea that all the students were building on was helping another student figure out how to make a sprite move in a circle.

These vignettes should give a sense of the flavor of the school. While some children are occupied with computers, regular school life goes on for others. The class next door may have five computers sitting idle while a geography or reading lesson is being presented much as it would be in any other school. Computers are accepted by the teachers and students as an integral part of the school, but they are not allowed to dominate it.

Some of the anticipated results of the Lamplighter Project have never happened. For example, the students have not used computers for creative writing, despite the availability of a simple screen editor as part of the Logo system. The equipment provided to the school in the first three years of the project has not included printers, which would be needed to make creative writing a realistic activity. Nor has Logo been integrated into as much of the school's curriculum as had been planned. According to Lamplighter's headmistress Pat Mattingly, "The teachers just don't have enough time for curriculum development in addition to all their other duties." With a few minor exceptions, the research studies that were expected to be part of the project have not materialized. Some unique, exciting, and wonderful things have been happening at the Lamplighter School, but except for the school staff, who usually are too busy to write, study, and reflect on the situation, one gets the feeling that "nobody's watching."

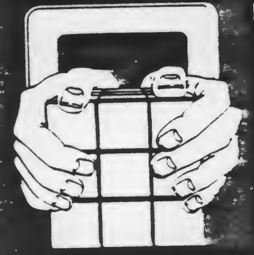
Other Interesting Projects

To round out the picture, I want to mention some other schools at which Logo is being used for research and development. The Cotting School for the physically handicapped in Boston has been the site of a series of projects conducted by Dr. Sylvia Weir of MIT. In these projects, Logo has enabled students with cerebral palsy, previously unable to communicate effectively, to begin to realize their intellectual potential. Of all the Logo projects, this has been the most dramatic in demonstrating Logo's effectiveness for students who previously had not been successful in academic settings. It has also made the most significant progress toward the goal of finding objective ways of determining just what students learn as they engage in Logo activities.

Another Logo project aimed at discovering what students are learning is being conducted by the Center for Children and Technology of Bank Street College in New York City. In this project, students in grades three through six have extensive access to Logo. The research is focusing on students learning problem-solving techniques and on social interaction among students as they work on Logo activities—two areas that have been highlighted, but not carefully studied by other projects.

TM

The Programmable Cube is like the popular physical game: it comes in many colors, you can turn its faces and rotate it, you can scramble it and solve it by moving the faces. It even comes with a complete solver for any scrambled cube. But it has an extra twist: it speaks a language especially designed for cubes through which you can write your own cube programs, either to produce fancy patterns or even to solve a scrambled cube with your own cube-solving strategy.



- High resolution graphics, color or black and white, shows all 6 sides and simulates motion.
- Tell it about your scrambled cube—it will lead you through the solution.
- Write your own cube programs: comes with complete programming environment including editor and debugger.
- Available for Apple II® or Apple II Plus®, 48K or 64K.

Apple is a Trademark of Apple Computer, Inc.

How to Order:
Send $34.95,
Check, VISA or
MasterCard to:

TM

Specify 48K or 64K.

P.O. Box 31337, Hartford, CT 06103

Logo Information Sources

Here is a partial listing of organizations that offer Logo training and information:

1. Logo Training Courses and Workshops:

Austin College, Sherman, TX 75090. Contact Prof. Henry Gorman.

Bank Street College, 610 West 112th St., New York, NY 10025. Contact Karen Scheingold.

Lesley College, 29 Everett St., Cambridge, MA 02138. Contact Nancy Roberts.

Logo: The Learning Center, Logo Computer Systems Inc., 989 Avenue of the Americas, New York, NY 10018. Contact Mike Tempel.

Teachers College, Columbia University, Microcomputer Resource Center, 525 West 120th St., New York, NY 10027. Contact Karen Billings.

Technical Education Research Centers, 8 Eliot St., Cambridge, MA 02138. Contact Robert Tinker.

University of Wisconsin—Oshkosh, Microcomputer Applications Group, Oshkosh, WI 54901. Contact Don Voils.

2. Organizations, Users' Groups, and Newsletters:

Boston Computer Society Logo Users Group, One Center Plaza, Boston, MA 02108.

FOLLK, Friends of LISP/Logo and Kids, 436 Arballo Dr., San Francisco, CA 94132.

Friends of the Turtle, POB 1317, Los Altos, CA 94022.

Logo Times, included in 99'er Magazine, POB 5537, Eugene, OR 97405.

LOGOPHILE, Logo Special Interest Group, c/o Higginson, Faculty of Education, Queens University, London, Ontario, K7L 3N6 Canada.

Monadnock Area Logo User's Group, c/o Dan and Molly Watt, Gregg Lake Rd., Antrim, NH 03440.

Young People's Logo Association, 1208 Hillsdale Dr., Richardson, TX 75081.

A third interesting Logo school project is not a research project at all. At Lincoln-Sudbury Regional High School in Sudbury, Massachusetts, students learn Logo as the introductory computer programming language. Experiences at Lincoln-Sudbury may show the way to those seeking to use Logo with older students.

Conclusions

I will take the risk of drawing a few general conclusions from these very diverse projects.

Logo can be effective for all students in a school setting. In fact, a regular theme of all the projects cited is the success of students who previously had been unsuccessful in school.

Teacher training is critical. At the very least, teachers need to understand the value of exploratory learning and student interaction. Further, at all sites involving Logo in classrooms, teachers have felt the need for continued support and training. While this need may diminish as teachers become more familiar with computers and Logo, it seems to be a reality for the present.

Teachers and students need resource materials, guidebooks, project suggestions, etc. The more specific the goals, as in the Edinburgh Logo Project or in the physics activities of the Brookline Logo Project, the more specialized and extensive the materials needed.

Student interaction has been a critical and positive element of all classroom-based Logo projects. In each case, students have taken on significant roles as teachers of other students, even as teachers of their own teachers.

In no case has the "full potential of what might be possible" with Logo been realized. It will probably take a lot of time, and many diverse efforts, before the learning potential of Logo can be fully understood and utilized. Whether the goal is to integrate Logo into existing school subjects or to use

IBM
personal computer
UCSD
p-System™ IV.1

A superior implementation of the UCSD p-SYSTEM is available NOW for your IBM Personal Computer.
Network Consulting has added many features that the professional programmer will find useful. These features are not available from any other sources.

- Hard disk support for Corvus, Tall Grass Technology and others
- 25% more floppy storage, without adding or modifying hardware, retaining compatibility with standard IBM diskettes
- Double sided and double sided/double tracking stepping floppy disk drive support
- Up to 800K on each mini-diskette
- RAM disk support (a pseudo-floppy volume that uses up to 512k RAM)
- 8086/87/88 macro assembler
- Extended memory support
- Adaptable system support available for adding custom I/O drivers
- Remote Terminal support
- XON/XOFF flow control for serial printers
- A library of program modules and other utilities
- Disk write verification
- and more!

Standard p-System features include:

- Standard I/O redirection (including command files)
- Dynamic program overlays
- Support for asynchronous processes and concurrency primitives in Pascal
- Fast Pascal p-code compiler
- Fast FORTRAN and Basic p-Code compilers also available
- A powerful screen oriented editor
- A filer for handling both files and volumes
- A Native Code Generator that processes your Pascal, Fortran and Basic programs to convert parts of them to native machine code
- Dynamic runtime binding of separately compiled programs and units
- Turtlegraphics for easy graphics displays
- Print spooler for background printing
- Symbolic debugger

USCD p-SYSTEM from Network Consulting
The only serious choice for business

TM The Regents of the University of California

MasterCard VISA AMERICAN EXPRESS

Network Consulting Inc.
A106 – 1093 W. Broadway
Vancouver, B.C.
Canada V6H 1E2
(604) 738-3500

Logo to develop entirely new kinds of learning environments, much work remains to be done.

During the past year, the use of Logo in schools has jumped from less than a dozen sites to hundreds. By the end of the coming year, it may involve thousands of classrooms with tens of thousands of students. As we struggle with the task of integrating new forms of learning into old structures, we should be particularly aware of the opportunity to learn from each other and from the limited, but carefully supported, research and development that have already occurred. ■

References

1. The Edinburgh Logo Project, Department of Artificial Intelligence, University of Edinburgh, Forrest Hill, Edinburgh, EH1 2QL Scotland.
   du Boulay, B. "Learning Teaching Mathematics." *Mathematics Teaching*, No. 78, March 1977.
   du Boulay, B. *Teaching Teachers Mathematics Through Programming*. DAI Research Paper No. 113, 1979.
   du Boulay, B. and T. O'Shea. *How to Work the LOGO Machine: A Primer for ELOGO*. DAI Occasional Paper No. 4, 1976.
   Howe, J. A. M. and T. O'Shea. "Learning Mathematics Through Logo." *ACM SIGCUE Bulletin*, Vol. 12, No. 1, January 1978.
   Howe, J. A. M., T. O'Shea, and F. Plane. *Teaching Mathematics Through Logo Programming: An Evaluation Study*. DAI Research Paper No. 115, September 1979.
2. The Brookline Logo Project, MIT Logo Group, Building 20C, Room 109, Massachusetts Institute of Technology, Cambridge, MA 02139.
   Bibliography of Logo Memos. MIT Logo Group.
   Papert, Seymour, et al. *Assessment and Documentation of a Children's Computer Laboratory*. Logo Memo 48, MIT Logo Group, 1977.
   Papert, Seymour, Andrea diSessa, Daniel Watt, and Sylvia Weir. *Final Report of the Brookline Logo Project: Project Summary and Data Analysis*. Logo Memo 53, MIT Logo Group, 1979.
   Watt, Daniel. *Final Report of the Brookline Logo Project: Profiles of Individual Student Work*. Logo Memo 54, MIT Logo Group, 1979.
   Watt, Daniel. *A Comparison of the Problem Solving Styles of Two Students Learning Logo*. Proceedings of the National Educational Computing Conference, 1979. (Reprinted in *Creative Computing*, December 1979.)
   Watt, Daniel and Sylvia Weir. "Logo: A Computer Environment for Learning Disabled Students." *The Computing Teacher*, Vol. 8, No. 5, May 1981.
3. Computers in the Schools, Bonnie Brownstein, Director, New York Academy of Sciences, 2 East 63rd St., New York, NY 10021.
4. Lamplighter School, Headmistress Pat Mattingly, 11611 Inwood Rd., Dallas, TX 75229.
5. Logo for Handicapped Students.
   Goldenberg, E. Paul. *Special Technology for Special Children*. Baltimore: University Park Press, 1979.
   Papert, Seymour and Sylvia Weir. *Information Prosthetics for the Handicapped*. Logo Memo 51, MIT Logo Group, 1978.
   Weir, Sylvia. *Evaluation and Cultivation of Spatial and Linguistic Abilities in Individuals with Cerebral Palsy*. Logo Memo 55, MIT Logo Group, 1980.
6. Center for Children and Technology, Bank Street College, Karen Scheingold, Director, 610 West 112th St., New York, NY 10025.

◀ **.5 MB STORAGE**

▶ **1.2 MB STORAGE**

We're offering you our SB-80 system in either 5 1/4" or 8" disk drives, your choice. Either way your system comes with a full size (12" diagonal) non-glare tiltable green screen with 24 lines by 80 character format. Its multi-character set offers blinking cursor, underlining, reverse video, and half and zero intensity. The movable, detachable keyboard has a numeric pad with cursor control and function keys.

■ Single Board Technology ■ CP/M® Operating System
■ 4 MHz Z80A CPU ■ 64K 200ns Main Memory
■ 8-Inch Dual Density Floppy Drives
■ 5 1/4-Inch Dual Density Floppy Drives
■ 2-Serial Ports ■ 2-Parallel Ports
■ 4-Counter/Timers ■ Expandable

For further information about this limited offer call or write:

Nationwide on-site and depot repair service through the professionals at INDESERV.

®CP/M is a registered trademark of Digital Research, Inc.

Colonial Data Services Corp., 105 Sanford Street, Hamden, Conn. 06514 • (203) 288-2524 • Telex: 956014

Designing Computer-Based Microworlds

Well-designed Logo procedures can help children grasp ideas of intrinsic interest.

R. W. Lawler
Le Centre Mondial L'Informatique et Resources Humaine
24 Rue Clemente Marot
Paris, France 75008

Designing computer applications for education might be called cogni[illegible]e engineering, for its objective is to shape children's minds. That lofty goal must carry with it a commitment to cognitive science, the study of how knowledge functions and changes in the mind. In light of the profound influence of computers in the schools, designing educational applications without such a commitment would be irresponsible.

I believe that Jean Piaget, the Swiss student of knowledge, formulated the general solution to the problem of how intelligence develops. Although the field of cognitive science has advanced beyond Piaget's innovative theories by revising and extending them, his insights into the nature of learning continue to influence teaching methods. The union of computer microworlds and Piagetian theory is the subject of this article.

Piaget and Education

Central to the work of Piaget is constructivism, the view that the mind incorporates a natural growth of knowledge and that the mind's structure and organization are shaped by interactions among the mind's parts. In *The Science of Education and the Psychology of the Child* (The Viking Press, 1971), Piaget challenges educators to answer two questions: How does instruction affect what is in the mind? and What remains in the mind from the process of instruction long after the time of instruction has passed? In the same work, Piaget disputes both the effectiveness and the ethical correctness of many of the practices of modern education:

If we desire to form individuals capable of inventive thought and of helping the society of tomorrow to achieve progress, then it is clear that an education which is an active discovery of reality is superior to one that consists merely in providing the young with ready-made wills to will with and ready-made truths to know with.

The Dilemma of Instruction

Given Piaget's view that learning is a primary, natural function of the healthy mind, we might consider instruction in any narrow sense unnecessary. Children (and older students of life as well) learn the lessons of the world, effectively if not cheerfully, because reality is the medium through which important objectives are reached. Nevertheless, in certain situations children often rebel against the lessons society says they must learn. Thus the educator's ideal of inspiring and nurturing the love of learning frequently is reduced to motivating indifferent or reluctant students to learn what full functioning in our society requires.

Teachers face a dilemma when they try to move children to do schoolwork that is not intrinsically interesting. Children must be induced to undertake the work either by promise of reward or threat of punishment, and in neither case do they focus on the material to be learned. In this sense the work is construed as a bad thing, an obstacle blocking the way to reward or a reason for punishment. Kurt Lewin explores this dilemma in "The Psychological Situations of Reward and Punishment" (*A Dynamic Theory of Personality: Selected Papers of Kurt Lewin*, McGraw-Hill, 1935). The ideas of Piaget and Lewin have led me to state the central problem of education thus: How can we instruct while respecting the self-constructive character of mind?

Computer-Based Microworlds

In *Mindstorms: Children, Computers, and Powerful Ideas* (Basic Books, 1980) Seymour Papert pro-

STATUS QUO Presents SPECIAL PRICE$

[illegible]ISK DRIVES	YOUR COST
[illegible]P Hard Disk Drive 10 Mb Formated	$3200
SHUGART 851R Dbl Dens Dbl sided 8"	495
APPLE COMPATIBLE MICRO-SCI A2 35 track drive	425
APPLE COMPATIBLE MICRO-SCI A40 40 track drive	399
APPLE COMPATIBLE MICRO-SCI A 70 70 track drive	575
APPLE COMPATIBLE MICRO-SCI Controller for above	100
APPLE PRODUCTS	
MICROSOFT Premium Pack	575
MICROSOFT 16K Ram Card	140
MICROSOFT Z80 Card	299
WESPER 16K Ram Wizard Card	95
NOVATION CAT Modem	135
NOVATION D CAT Modem (direct RS-232 connection)	148
NOVATION APPLE CAT II	329
SUP'R'MOD II (converts TV to monitor)	33
RH Super Fan II	55
T.G. Products Joystick	53
SOFTWARE	
VISICORP's VISICALC for Apple II	150
MICRO PRO WordStar (Apple II)	195
MICRO PRO WordStar (CP/M)	249
MICRO PRO SuperSort (Apple II)	109
MICRO PRO SuperSort (CP/M)	125
MICRO PRO CalcStar (CP/M)	189
MICRO PRO CalcStar (Apple II)	139
MICRO PRO DataStar (CP/M)	219
MICRO PRO DataStar (Apple II)	199
STONEWARE DB Master	179
[illegible] Easy Writer 40 Column (Apple II)	89
[illegible]TON-TATE dBASE II (CP/M)	479
[illegible]SCI/SOFTAPE Magic Window (Apple II)	79
S-100 PRODUCTS	
CALIFORNIA COMPUTER SYSTEMS 2422-Floppy Ctrl CP/M 2.2	323
CALIFORNIA COMPUTER SYSTEMS 2710 4 serial I/O A & T	269
CALIFORNIA COMPUTER SYSTEMS 2065 64K Static Ram A & T	519
CALIFORNIA COMPUTER SYSTEMS 2032 32K Static Ram A & T	621
CALIFORNIA COMPUTER SYSTEMS 2116 16K Static Ram A & T	239
CALIFORNIA COMPUTER SYSTEMS 2720 4 parallel A & T	189
HAYES Micromodem 100	329
HAYES Smartmodem	217
HAYES Chronograph	189
PRINTERS & PRODUCTS	
EPSON MX-80 F/T	745
EPSON GRAFTRAX 80	73
MICROCOMPUTER & MONITORS	
NEC 8001A Microcomputer w/32K RAM	710
NEC 8012A I/O Unit w/32K, expansion slots	590
NEC 8031A Dual Mini-disk drive unit	710
AMDEK Video 100 (12" B/W)	129
AMDEK Video 300 (High Res Green)	219
SANYO 13" Color	490

CREDIT CARD ORDERS CALL
(800) 227-5414
M–F 9am–5pm PST

All prices are cash prices. Add 3% surcharge when using Master Card, Visa or American Express. California residents add 6% tax. For shipping and handling charges, please add $2 per item for drives and printers, and add $10 per item for microcomputer and monitor items. Prices are subject to change without notice.

Make all checks payable to: **STATUS QUO**

(We must allow two weeks for all personal checks to clear so for faster service, please send a money order or cashier's check or order using your credit card.)

STATUS QUO, div. of I.T.A.T. Associates, Inc., 2095 Van Ness Ave., San Francisco, CA 94109. For technical questions or special orders/prices call (415) 474-1007.

Circle 197 on inquiry card.

DEALERS: We purchase excess inventory. Please call us with prices.

poses computer-based microworlds as a general solution to the problem of motivation. One argument for Papert's proposal runs as follows: learning is often a gradual process of familiarization, of stumbling into puzzlements, and resolving them by proposing and testing simple hypotheses in which new problems resemble others already understood. Microworlds are in essence "task domains" or "problem spaces" designed for virtual, streamlined experience. These worlds encompass objects and processes that we can get to know and understand. The appropriation of the knowledge embodied in those experiences is made possible because the microworld does not focus on "problems" to be done but on "neat phenomena"—phenomena that are inherently interesting to observe and interact with.

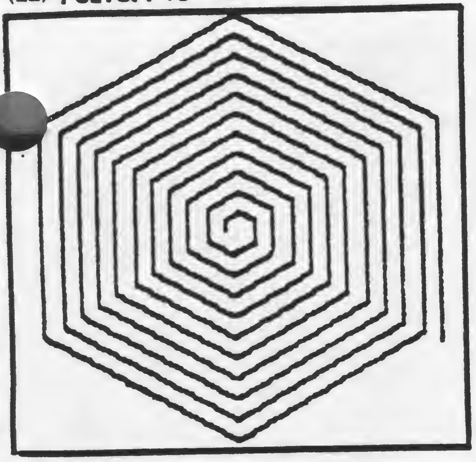
With neat phenomena, the challenge to the educator is to formulate so clear a presentation of their elements that even a child can grasp their essence. A well-designed computer microworld embodies the simplest model that an expert can imagine as an acceptable entry point to richer knowledge. If a microworld lacks neat phenomena, it provides no accessible power to justify the child's involvement. We can hardly expect children to learn from such experiences until they are personally engaged in other tasks that make the specific knowledge worthwhile as a tool for achieving some objective. This amounts to an appropriate shifting of accountability from students (who have always been criticized for not liking what they *must* learn) to teachers, those who believe that their values and ideas are worth perpetuating.

Computer-based microworlds help tailor instruction more closely to Piaget's idea of the natural mode of learning. I will illustrate this point by presenting two examples of computer-based microworlds.

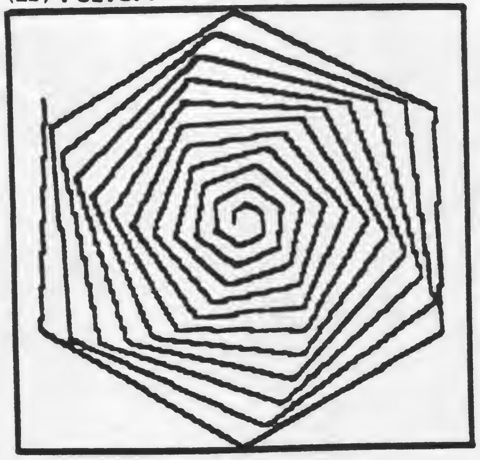
The POLYSPI Microworld

POLYSPI (from "polyspiral") is a name for a three-line procedure in the Logo language and for the class of de-

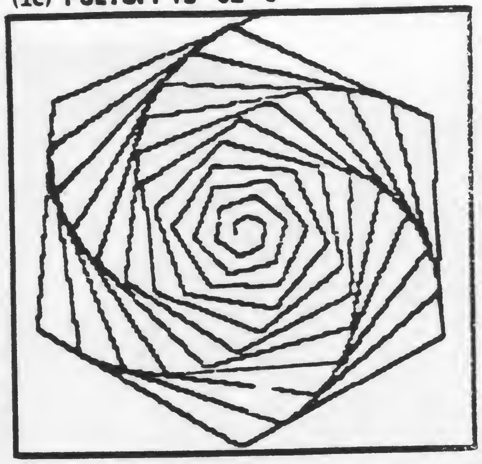
(1a) **POLYSPI 10 60 3**



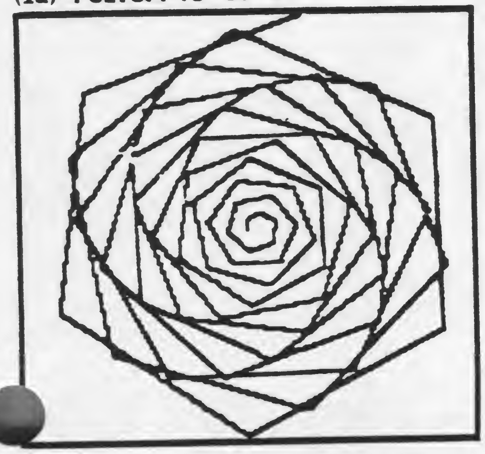
(1b) **POLYSPI 10 61 3**



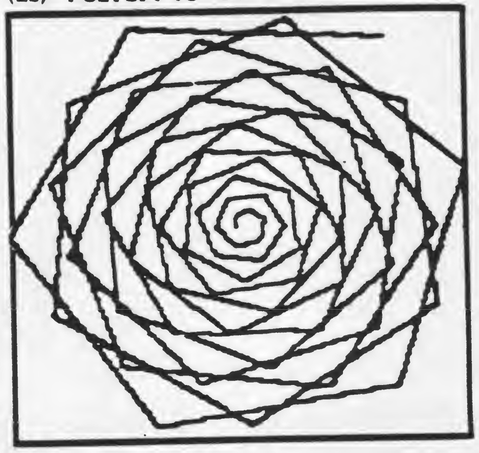
(1c) **POLYSPI 10 62 3**



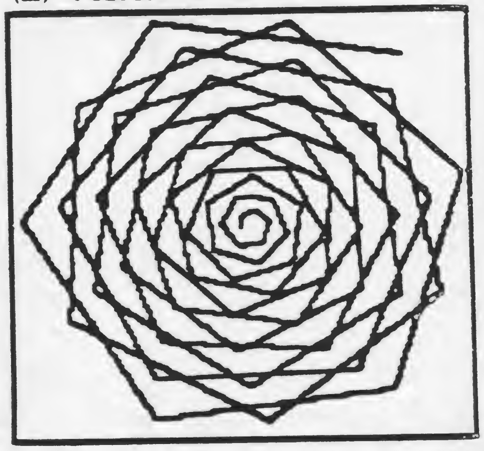
(1d) **POLYSPI 10 63 3**



(1e) **POLYSPI 10 64 3**



(1f) **POLYSPI 10 65 3**



Figures 1a–1f: *Polyspiral designs generated by changing one variable of the three in the* POLYSPI *procedure (shown in listing 1). The procedure's variables are* DISTANCE, ANGLE, *and* CHANGE *(in distance). The procedure draws a design by going forward the specified distance, turning at the specified angle, then increasing the distance by the specified change, going forward for the incremented distance at the specified angle, and so on. In this example, the distance variable and the change in distance are held constant. The angle variable is stepped up by one degree in each design. The strikingly different designs show the power of the concept of stepping variables.*

signs produced by different executions of that procedure. Figures 1a–1f show some examples of POLYSPI designs. The POLYSPI procedure is stated in listing 1. Some of the designs are pretty, mainly because surprising spiral patterns emerge under certain conditions. The general appeal of POLYSPI designs largely accounts for the adoption of turtle graphics as a subsystem of languages such as Smalltalk, Pascal, and even some implementations of PILOT. The variability of the POLYSPI procedure sometimes permits even a beginner to surprise more expert users (as well as himself) with the discovery of beautiful designs.

The procedure in listing 1 and its designs comprise a microworld. The objects of the microworld are all the designs that the procedure can generate, an engaging and extensive domain for exploration. More important, the designs are a class of "neat phenomena" whose generation can be made comprehensible with the following small set of ideas. First, the POLYSPI procedure provides a crisp model of variable separation: the three variables DISTANCE, ANGLE, and CHANGE are each used once, and used differently, in a simple procedure text. Second, the difference in relative potency of the variables (the impact of a unit change on the produced design) is obvious and striking. (ANGLE and then CHANGE are much more potent than DISTANCE.)

The POLYSPI microworld reveals the powerful idea of stepping variables.

The POLYSPI microworld reveals the stepping of variables as a powerful idea. By stepping variables I mean identifying one variable as a dimension of examination and holding all other variables constant while the chosen one is varied incrementally. In short, this microworld provides a clear model of how particular things may be generated through their intersecting dimensions of variation. Piaget judged variable-stepping to be an essential component of formal operational thought. The idea is a powerful one because it is almost universally useful; it is crucial to the process of scientific investigation.

Within the microworlds of turtle geometry, the insights achieved with POLYSPI exploration are easily extended to a related microworld of INSPI designs. The INSPI procedure differs from POLYSPI only in that the

Listing 1: *The* POLYSPI *procedure, written in Logo. From only three variables—distance, angle, and change in distance—this procedure can generate a remarkable variety of polyspiral designs. The procedure draws by going forward the specified distance, turning at the specified angle, then increasing the distance by the specified change, going forward for the newly incremented distance at the same specified angle, ans so on. Some designs drawn by* POLYSPI *appear in figures 1a–1f.*

```
TO POLYSPI :DISTANCE :ANGLE :CHANGE
FORWARD :DISTANCE
RIGHT :ANGLE
MAKE "DISTANCE :DISTANCE + :CHANGE
POLYSPI :DISTANCE :ANGLE :CHANGE
END
```

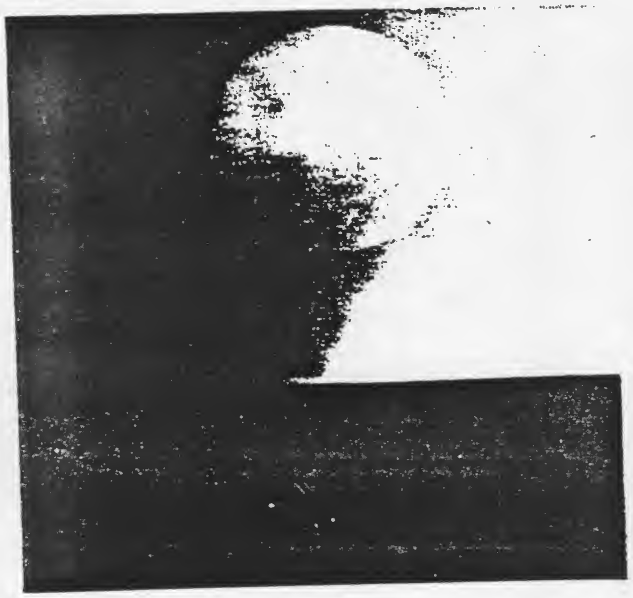
change value is applied to the ANGLE variable instead of to the DISTANCE variable. (For a case study of a child's ability to grasp and extend this idea, see my article "Extending a Powerful Idea," in a forthcoming issue of *The Journal of Mathematical Behavior*.)

The BEACH Microworld

The adolescent's initiation to formal thought differs greatly from the preschooler's introduction to reading, yet both learning experiences involve grasping central representations. What the prereader learns in an alphabetic language is a serial symbolic representation for words that signify the names of objects, actions, and so on. Let me here describe a Logo microworld for learning the alphabetic language. This microworld helped my 3-year-old daughter learn to read with minimal direct instruction.

While previous Logo implementations focused on a single, all-important agent—the turtle—TI Logo also has sprites. A sprite is a video-display object that has a location, a heading, and a velocity, but no drawing capability. It may be associated with a shape (which it "carries" and which assumes one of 16 colors). The shapes can be easily defined and changed by the Logo user. There may be a maximum of 25 shapes. The importance of a multitude of easily discriminated objects for early language applications cannot be overestimated. TI Logo has a second graphics system, "tile graphics," that is compatible with the sprite graphics system. The static tiles, which may also assume 16 different colors and exhibit modifiable shapes, provide a suitable "background" for the movements of the dynamic sprites. The result is the opportunity to create scenarios that have many moving objects with different shapes and different colors and a static but vivid backdrop. The BEACH microworld permits the creation of such scenarios, as the scenes in photos 1a and 1b illustrate.

Allenbach is putting tomorrow on disks.

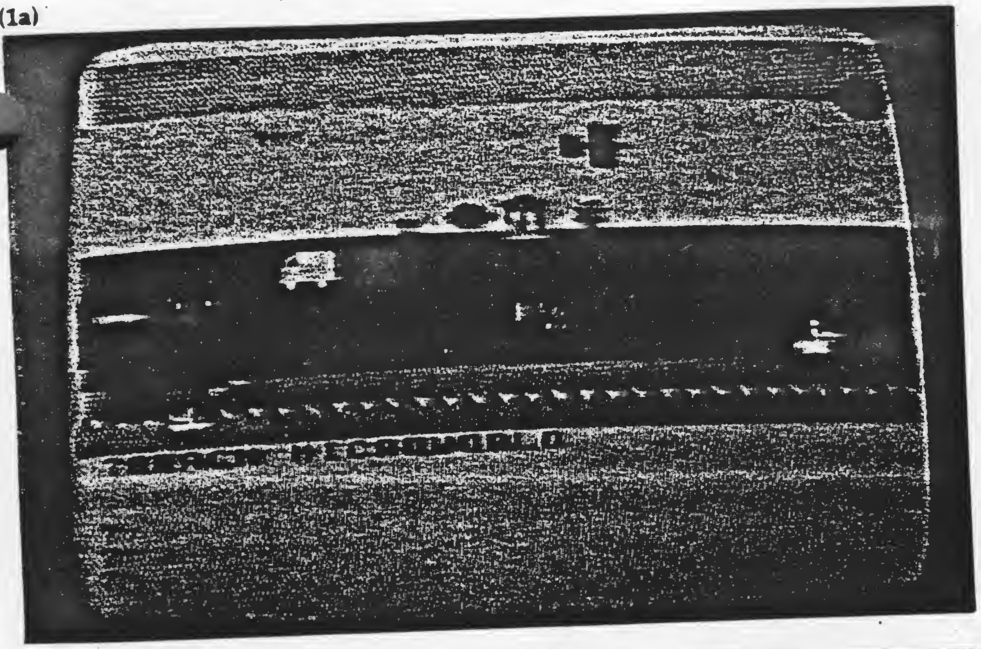


The future of the world is written in computer software. And Allenbach disk duplication is making sure that the future is faithfully reproduced. We use our own Allenbach diskettes and state of the art equipment, with electronic surface evaluation of each diskette before use. Copies are 100% verified, track by track, and carry the Allenbach lifetime warranty against defects in workmanship or materials.
For information and prices on disks, duplication, and dealer information, please call (800) 854-1515. In California, call (714) 438-2258 collect.

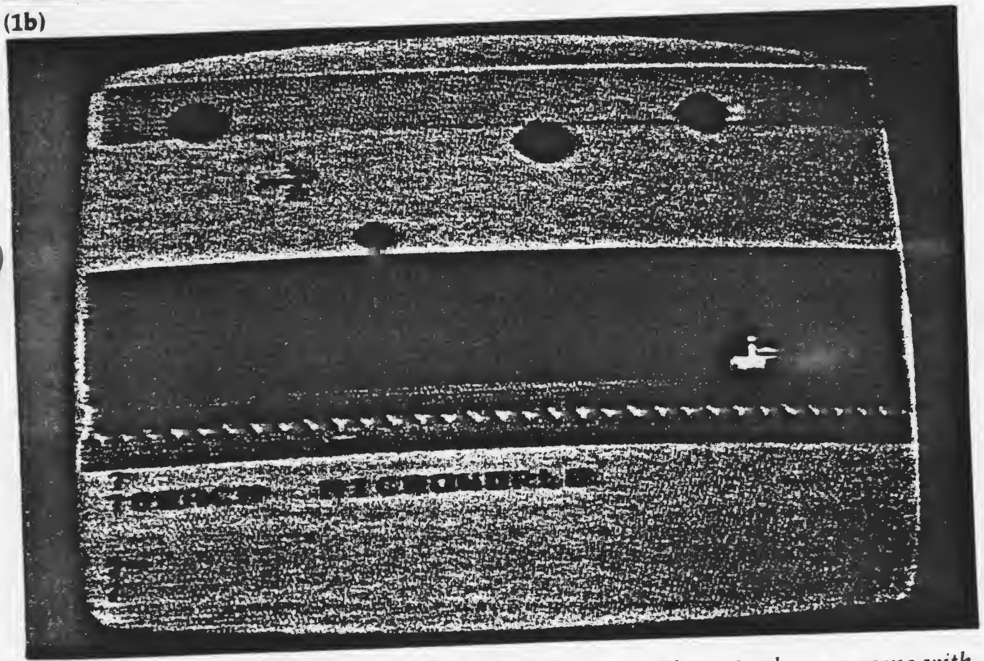
Allenbach Industries Inc.

Circle 12 on inquiry card.

(1a)



(1b)



Photos 1a–1b: *Two scenes from the* BEACH *microworld. Photo 1a shows a scene with many objects. Photo 1b shows a scene typical of those drawn by a 3-year-old child. The author's daughter learned to read 30 words by exploring the* BEACH *microworld, which the author and his children created using TI Logo with the Texas Instruments 99/4A microcomputer.*

Meaningful Names Ease Learning

Because Logo gives the user great freedom to define and name procedures, appropriate descriptive English words can be used. For example, SUN can be the name of the procedure that creates a yellow ball on the display. The word UP can name the command that increases the value of a sprite's *y* coordinate. Repeated often enough, UP puts the SUN in the sky above the BEACH. Another word, such as SLOW or FAST, can set the SUN in motion. Because new procedures are easily defined, the child, a family member, friend, or teacher could even make the SUN ZOOM if the child wishes. Such flexibility permits the microworld to be tailor-made to suit any child. To the extent that the child participates in defining the objects to be part of the world, their attributes, and the actions they are to perform, the microworld is also constructed *by* the child herself. My 3-year-old, Peggy, her older siblings, and I chose about 20 objects to populate her world, designed and made shapes to represent them, and wrote the procedures to create and manipulate them. The vocabulary of her BEACH world includes the following:

OBJECTS

BEACH, BIRD, BOAT, BOY, CAR, DOG, FISH, GIRL, HOUSE, JET, KID, MAN, MOON, OAK, PINE, PLANE, PONY, STAR, SUN, TRUCK, VAN, WAGON

ACTIONS

UP, DOWN, MOVE, BACKWARD, FAST, FLY, HALT, SAIL, SHOW, SWIM, TURN, WALK, ZAP, ZOOM, PAINT BLACK, PAINT GREEN, etc.

When Peggy began to play with this computer microworld, she did not recognize any words except "by," and she had no idea what that meant. Her ability to discriminate between letters and name them was undependable and idiosyncratic. She began keying words, copying them letter by letter from a set of 4- by 6-inch cards I had made. Soon she began keying her favorite or most-used words from memory, and later she was able to read those individual words in other contexts. Now she deals with the written language one word at a time (as infants begin to speak with specific signification). To handle sentences (other than "paint some-color-name") or begin phonetic decoding of words, she will need more complex microworlds.

For Peggy, the learning of reading and the learning of writing have been synchronized (as speaking and interpreting speech are for the toddler); she learned to read her 30-word vocabulary by learning first to "write," i.e., key the words on the computer terminal. Writing was an essential part of controlling the computer microworld that engaged her. My role as teacher changed from taskmaster to occasional consultant. I would answer questions Peggy brought me after she had tried to work with the constructed reality of the BEACH microworld, and I helped her when she had problems, but I offered her no lessons beyond the rule that words are keyed letter by letter,

Circle 169 on inquiry card.

ERG/68000
MINI-SYSTEMS

☐ Full IEEE 696/S100 compatibility

HARDWARE OPTIONS

☐ 8MHz or 10 MHz 68000 CPU
☐ 4-8 RS232 Serial Ports
☐ 64K STATIC RAM (70 nsec)
☐ 256K Dynamic RAM, with full parity
☐ 8" D/D, D/S floppy disk drives
☐ 5MB-32MB hard disk drives
☐ Full DMA host adaptor
☐ 20MB tape streamer
☐ 10 to 20 slot backplane
☐ 30 amp power supply

SOFTWARE OPTIONS

☐ 68KFORTH[1] systems language with MACRO assembler
☐ Fast Floating Point package
☐ Motorola's MACSBUG
☐ IDRIS[2] operating system with C, PASCAL, FORTRAN 77, 68K-BASIC[1] compilers

Trademark [1]ERG, Inc.
[2]Whitesmiths

30 day delivery
with valid Purchase Order

OEM prices available
For CPU, Integrated Card Sets
or Systems.

Empirical Research Group, Inc.
P.O. Box 1176
Milton, WA 98354
206-631-4855

left to right, and that a specific symbol meant she should press the Enter key.

I make no claim that computer microworlds can teach all reading skills, nor that this specific BEACH world would appeal to other children in different circumstances. I do, however, see the BEACH microworld as a prototype of the various worlds that others may fashion for small children.

Design Heuristics: Powerful Ideas

A computer microworld should be constructed around a powerful idea, one worth the instructor's time to develop or the student's time to explore. Who decides if an idea is sufficiently powerful? You do, at first, when you design a microworld. Next, the students determine the worth of the microworld as they incorporate the idea into their minds or reject it.

If you need a little guidance when you design a computer-based microworld, Papert (in *Mindstorms*) offers four criteria for powerful ideas: they should be simple, general, useful, and syntonic. The idea behind a microworld must be formulated as simply as possible; an idea can be powerful only when understood. Even if an idea is embodied in a specific microworld, it will not be useful through extension unless it is general.

Reality dictates the candidates for powerful ideas. Society also declares what ideas are important: if you can't read, for example, a technological society relegates you to subhuman status. But it is your own mind, more than any advice, that can tell you what ideas are powerful. Your own insights enable you to integrate important experiences. An idea is powerful, then, if it gives form to your understanding of life. It follows that you cannot inspire others with an idea unless it has first inspired you.

Interconnection of Knowledge

What Papert labels the "syntonic" characteristic focuses on how an idea assumes power within the mind of an individual. An idea is powerful for a person if it relates and unifies knowledge gained in diverse experiences. An idea gains power if it can be reduced to a concrete model that serves as a metaphor for the interpretation of subsequent problems. Such a model helps explain which aspects of new problems must be considered, which may be neglected, and which anomalies must be explained away on a basis of local evidence. Models prove more or less powerful depending on the individual's interests and experiences.

The most essential characteristic of powerful ideas is their relation to the individual's previous knowledge. You can tell students that one situation resembles another, but recognition of such comparisons is more powerful if it is the students' own discovery. They will make the connections between the structures of one idea and another at a level of detail appropriate to their specific prior knowledge and feelings.This internalization is the basis of an idea's power for the individual.

An analogy may help here. If you solder a connection at too low a temperature, you can get mechanical binding but undependable electrical contact. Ideas imposed by instruction are like badly soldered joints. Only the individual has the power to fuse connections between new ideas and his or her own most personal thoughts and feelings. These connections alone can make an idea an important part of how the person sees the world and behaves in it.

Paradoxically, an excellent way to harness the students' understanding for engagement with ideas is to liberate their expressiveness. Because Logo is a vehicle for free exploration, knowledge built from Logo is syntonic, appropriate to the person, and experienced as an authentic, intimate part of the self. Such is the power of an approach to learning that frees the individual to create within a social context that makes our culture's most powerful ideas accessible.

I/O and Applications Design

An application design negotiates between a specific objective and the potential of the equipment. Computers are general-purpose symbol manipulators, so they can deal abstractly with an idea. What any computer system can do in an interesting way, however, depends on its input/output (I/O) devices. Look for something special about a machine's Logo to suggest the kinds of neat phenomena the system could exhibit. Consider these examples from previously implemented Logo systems:

- The accessibility of the robot floor-turtle world to a child's physical intervention can lead even a small child into simulating the turtle's actions and into debugging procedures (after fixing a procedure "manually," a child can become more engaged in fixing it symbolically).
- Turtle graphics—whose appeal depends largely on the emergence of patterns from simple procedures that command the drawing of many lines—came into its own only with the general availability of bit-map-based displays.
- Logo on the GTI-3500 had a significant potential for engineering and physics simulations because a hardware-implemented "spin" primitive extended the forward and right primitives of "classical" Logo.
- The TI 99/4A joins together a general-purpose microprocessor (where TI Logo is implemented) with a special-purpose graphics processor that manipulates the sprites that give TI Logo its most striking effects.

As increasingly powerful microprocessors become affordable, the special quality of each will bring new potential for creating engaging microworlds. More powerful microprocessors and graphics slave processors may, for example, bring molecule modeling within reach. Local networks of small machines may permit group simulation of economic and political situations (as in games) that are now too abstract, rule-driven, and theoretical to interest many young people. There will continue to be opportunities for creating microworlds around the most powerful areas of contemporary science and technology.

Objects in Microworlds

Logo procedures can serve as a [illegible] formal systems. The commands of Logo are designed to communicate with a computer and its output devices, but the extension of Logo through procedures whose names are natural-language words can make the objects and actions more comprehensible. This ability to be extended is a key feature for young children.

But Logo is only a quasi-natural language; a Logo procedure must run on a machine. Further, the objects of a Logo microworld are formal; they can be completely defined by a specification of their state variables. One of the simplest of these objects is the Logo screen turtle. Once you have specified the turtle's location, heading, and pen position, there is no more to say about it. The operations of a microworld are also completely specifiable in terms of the effects they have on state variables. The RIGHT and LEFT commands, for example, modify the heading of the turtle but do not affect its location. Given the ease of specifying the interaction of state-change operations with state variables, a first criterion for the quality of any Logo implementation (an application microworld or the interpreter itself) is the clear presentation of the state variables to someone using the system. Two examples of representation inadequacies in TI Logo can clarify the point: although the heading of a sprite is a significant state variable, it cannot be determined by inspecting the object's appearance (the shape carried by the sprite) when its velocity is zero; it is impossible to determine visually which sprite is the "current" object, i.e., the one or ones that will respond to the next Logo command. Ideally, the equivalent of a SHOWTURTLE/HIDETURTLE set of commands would show which is the active sprite. Whatever the limitations of a specific Logo implementation, anyone who designs a computer-based microworld should strive to represent all the state vari-

[illegible]

[illegible] Turnaround time in hours and days, not weeks. You get your disks when you need them.

[illegible] Each copied track is checked bit by bit. If it doesn't match your original, it isn't shipped.

[illegible] Standard, double-boot, and copy-resistant formats. 50 copies minimum, no maximum.

[illegible] Set-up as low as $10 per disk. Copying only from 30¢, copying and 3M disk from $2.00. Other disks and services also available.

[illegible] Years of disk duplication experience insures a problem-free shipment on time — time after time.

For all your Apple® compatible copying needs.

ALF 1448 Estes Denver, CO 80215 (303) 234-0871 ables in a visible, obvious way. Doing so enhances the comprehensibility of the ideas embodied in the objects and their manipulations.

Comments on a New Microworld

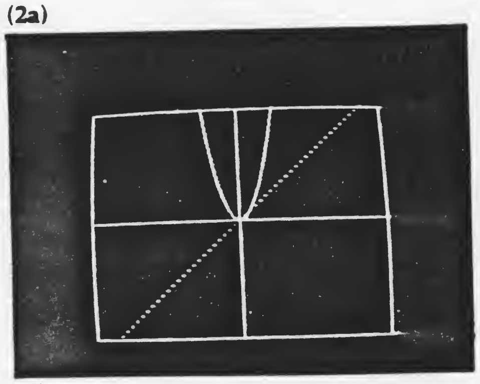
One of the objectives of Logo is to put power in the hands of beginning users. Even powerful ideas usually come from striving to reach a simple, down-to-earth objective. To demystify designing a microworld, I would like to present a few comments on some work in process. I wanted to develop an effective way to present some ideas of algebra to a 12-year-old. I remembered a casual comment of a former MIT Logo colleague, Andy diSessa, that one of the most powerful ideas accessible through Logo was embodied in "procedures that output." At the time, I was mystified, even though Andy had explained that his comment was based on the fact that such a procedure was equivalent to a mathematical function. *That* observation came back to me. Algebra is about mathematical functions. Although I couldn't fully appreciate Andy's comment, it focused my attention on a personally comprehensible way of expressing mathematical functions in Logo.

Common mathematical functions assign the value of one variable (call it y) to the value of some expression based on another variable (call it x). Assigning values is just what the MAKE command does. If a superprocedure controlled assigning to y the value of an x-based expression for the domain of possible values of x, it would generate any function expressible in the Logo language. When given two inputs (x,y), the DOT primitive of Apple Logo draws a dot at the screen location of those coordinate values. If the value of x is incremented across the domain of possible x-coordinate values, and y is specified in terms of the value of x, DOT can be used to plot discrete sketches of mathematical functions. A second method of drawing functions is possible. If those "dotted" locations are used as the position coordinates of a SETTURTLE (SETPOSITION) command, the screen turtle will draw a line-segment approximation to a mathematical function. These are the ideas around which the PLOTTING microworld is constructed. Photos 2a and 2b show examples from the PLOTTING microworld.

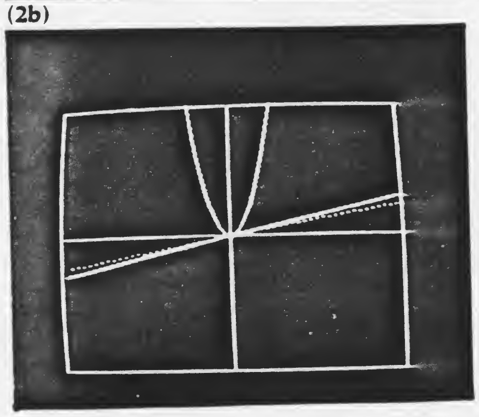
How can a person go from common experiences to a new idea by doing something only slightly unusual—but with that small difference providing access to a range of significant phenomena? Think about what kinds of experiences younger students might have had that could support learning about mathematical functions. Any child who uses Logo for a while learns to define specific variable values using the MAKE primitive; for example:

```
MAKE "MY.NAME "BOB
MAKE "MY.AGE 42
```

(2a)



(2b)



Photos 2a, 2b: *Two examples of the* PLOTTING *microworld. Photo 2a shows the contrast between the straight-line plot of $y=x$ and the parabolic plot of $y=.2x^2$. Photo 2b contrasts the same parabolic plot with a plot of $y=.25x$ (segmented line), an attempt to fit the slope (heavy line) of the parabola at a point. See listing 2 on page 158 for the* DOTPLOT *procedure from the* PLOTTING *microworld.*

The minimal significant complication possible in the specification of a variable is to make its value depend on something else, such as keyboard input. It is common for beginners to write routines such as the greeting below for inclusion in some more ambitious program:

```
TO GREET
PRINT [WHAT'S YOUR NAME ?]
MAKE "WHO READWORD
   ;accept keyboard input
PRINT (SENTENCE
   [GLAD TO MEET YOU,] :WHO)
END
```

We can start with nonarithmetic examples of variables as functions of other variables. They can be simple or complex. Graphs of equations can be viewed as another, more specific form of a familiar kind of relation—a new representation for a familiar idea. The algebraic formulas with which we usually associate the graphs of equations are seen as another description of a correspondence relation, a description that is specific and limited, but very powerful.

Making clear the connection between concrete uses of programming variables and mathematical functions is one justification of a PLOTTING microworld. This idea is one I judge to be powerful. The programming needed to make a Logo PLOTTING subsystem is nearly trivial (see listing 2), but that is precisely the virtue of a powerful language: its expressiveness makes ideas and functions stand clear of accidental complications.

Extending the PLOTTING World

If we look beyond the simple plotting of functions, the intellectual extensions of such a microworld can be simple and striking. Consider these two possibilities. First, when the domain of x is specified with beginning, end, and increment or step-size (to control the grain of the plotted function), the slogan through which continuity is often expressed becomes an almost obvious consequence of the "dotted" representation: "you give me an *epsilon*, and I can give you a *delta* such that whenever the difference between successive values of x

Listing 2: *The* DOTPLOT *procedure, written in Logo. The procedure plots y as a function of x for values of x incremented by* CHANGE. *Resulting plots appear in photos 2a–2b.*

```
TO DOT.PLOT :CHANGE
MAKE "X -135
PRINT [MAKE "Y A FUNCTION OF :X]
MAKE "FUNCTION READLIST
LABEL "AGAIN IF :X > 135 [STOP]
RUN :FUNCTION
IF NOT OR :Y > 119 :Y < -119 [DOT SE :X :Y]
MAKE "X :X + :CHANGE
GO "AGAIN
END
```

Listing 3: *The* LJ1 *procedure, written in Logo. Using the variables* COEFF1, COEFF2, *and* CHANGE, *the procedure draws Lissajous figures like those shown in photos 3b, 3c, and 3e.*

```
TO LJ1 :COEFF1 :COEFF2 :CHANGE
MAKE "ANGLE 0
MAKE "FUN1 (SENTENCE [MAKE "X 100 * SIN :COEFF1 * :ANGLE])
MAKE "FUN2 (SENTENCE [MAKE "Y 100 * COS :COEFF2 * :ANGLE])
PENUP RUN :FUN1 RUN :FUN2 SETPOS (SENTENCE :X :Y) PENDOWN
LABEL "AGAIN MAKE "ANGLE :ANGLE + :CHANGE
RUN :FUN1 RUN :FUN2
SETPOS (SENTENCE :X :Y)
GO "AGAIN
END
```

is less than *delta*, the difference between successive values of *y* will be less than *epsilon*." Second, consider the implications for understanding the differential calculus. When plotting the value of a function, it is simple to save the value of the prior point-couple and calculate the slope of the function. This is an empirical form of differentiation. A microworld of plotting tools (whose activities could include plotting functions, the empirical derivation of slopes of those functions, and curve fitting—with the plotting tools—to those empirically derived slopes) could provide a body of practical experience about the relations between functions and their slopes. This experience, for which differential calculus will later provide a theory, will make the calculus easier to appreciate and assimilate.

These ideas may interest a math teacher or a psychologist, but would any child be interested in plotting mathematical functions? Are there any accessible neat phenomena? This is the most important final question the creator of every microworld must face. The concrete appeal of this microworld must be the creation of appealing (and possibly puzzling) graphic designs. The beauty of turtle-geometry designs derives from the use of repetition and variables in simple procedures. This observation suggests that we look at repeating functions such as those produced by the sine and cosine primitives. Photos 3a–3f show six designs made from combinations of sine and cosine functions. These designs, generically known as Lissajous figures, are my candidates for neat phenomena of the PLOTTING microworld. [**Editor's Note:** *Named for French physicist Jules Lissajous, each of these figures consists of the series of plane curves traced by an object that executes two mutually perpendicular harmonic motions. . . .* **P.L.**] The method of the procedure shown in listing 3 is to calculate a screen location with *x* as a sine function of an angle value and *y* as a cosine function of the same angle. The design is made when the turtle draws a line as it moves from one calculated location to the next one. The procedure is stopped manually.

Lissajous figures are similar to POLYSPI designs in general character because they are made of line segments that show natural classes or families of shapes and occasionally emerge as surprisingly beautiful. Like INSPI designs, they are somewhat mysterious to those who think more concretely than formally. Are they

$$ READ THIS IF YOU WANT TO SAVE MONEY $$
BEST HARDWARE AND SOFTWARE DEALS AVAILABLE!

APPLE BUSINESS SOFTWARE	
Visicalc	$189
Versaform	$299
Milestone	$255
Supercalc	$225
DB Master	$185
DB Master Util. Pak	$ 79
Pro Easywriter Combo	Call for Best Prices
Accounting Plus	
BPI Software	
dBASE II	
FMS-80, 81, 82	
VISI Series Software	
★ **THE LAST ONE** ★ The program that writes programs! Special Offer with purchase.	Call
APPLE PERIPHERALS	
Thunderclock	$125
Basis Encoder Board (U/L Case)	$109
Mountain Computer Prod.	Call
Videx Products	Call
Microsoft Products	Call
Corvus Products	
SVA Products	
★ **NEW PRODUCT LINES** ★	
IBM Software and Peripherals	Call for Best Prices
Atari Software	
TRS-80 Software	
★ **INTRO SPECIAL** ★	
Basis 108 Computer	**Call**
Videx Visicalc PreBoot	Call
★ **SPECIAL** ★ **MICROSOFT PREMIUM SYSTEM,** incl.	
Microsoft Z-80 Card	All for Only $625
Microsoft 16K Card	
Videx 80 Column Card	
Videx Soft-Switch	
Osborne CP/M Guide	
★ **DISKETTE SPECIALS** ★	
Maxell MD-1 (Box of 10)	$ 32
Maxell 8" (Box of 10)	$ 41
Verbatim 5¼"	$ 29
Verbatim 8"	$ 35
COMMUNICATIONS	
Hayes Micromodem II	$275
Hayes Smart Modem	$235
Hayes Chronograph	$225
Novation Apple Cat II	$335
Micro Courier	$210
Pro Z-Term	Call
PRINTERS & INTERFACES	
NEC Spinwriter 7710	$2350
Anadex DP-9501	$1295
Okidata Microline 83A	$745
CCS Interface Cards	Call
Qume and Diablo	Call
MICRO PRO	
Wordstar	$230
Supersort	$130
Mail Merger	$ 99
Data Star	$230
Spell Star	$130
Calc Star	$130
CP/M 8" Versions	Call
COMPUTER SYSTEMS	
Xerox 820 Computer	Call
Xerox Peripherals	Call
MONITORS	
NEC Hi-Res 12" Green	$170
NEC Hi-Res 9" Green	$165
NEC RGB 12" Color	$995
Sanyo Monitors	Cal
Amdek Monitors	Ca
MISCELLANEOUS	
Micro Sci Drives	Cal
16-Bit Apple Card	Ca
TG Joysticks	$ 45
TG Paddles	$ 30
ABT Keypads	$ 99
System Saver Fan	$ 77
Disk Storage Pages (10)	$ [illegible]

VISA master charge

Minimum order $100. Cashiers checks and money orders accepted. Add 3% for VISA or MC. Add 3% for shipping/insurance/handling. UPS. No COD. Prices subject to change. Call to verify. Order hours 9-6 PDT. Mon-Fri. Personal checks allow 20 days to clear. All products with full manufacturer's warranty, factory sealed. Calif. residents add 6% sales tax.

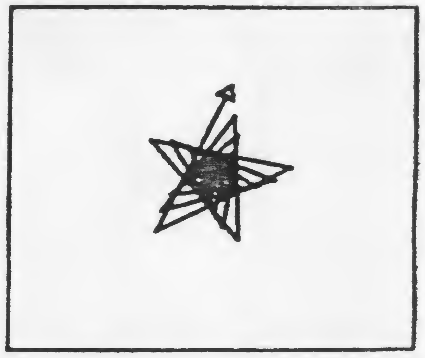
Apple is a registered trademark of Apple Computer Inc. All brand names are registered trademarks.

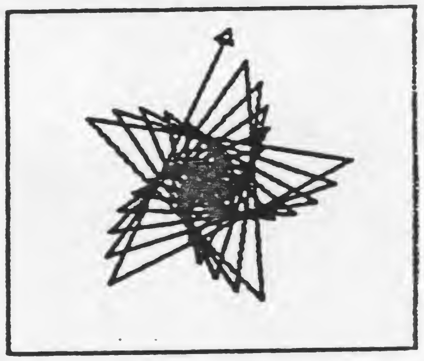
COMMUNI-CAL INC.™
1400 GRANT AVE. / NOVATO, CALIFORNIA 94947
IMMEDIATE DELIVERY
PHONE ORDERS CALL COLLECT
415-892-7139 or 415-897-1414

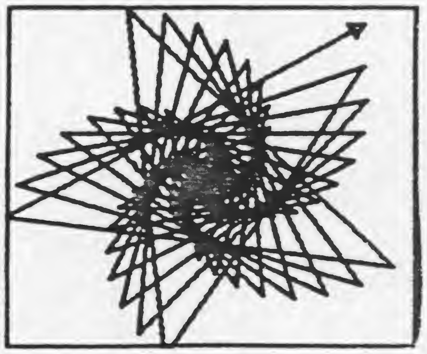
Circle 83 on inquiry card.

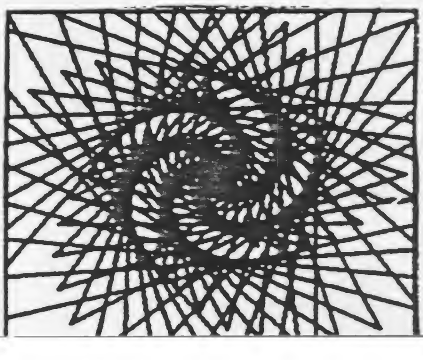
Apple Logo

Logo Computer Systems, Inc.







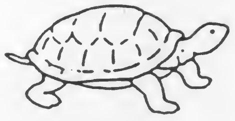


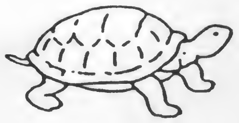
What is Logo?

Logo is a computer programming language, an educational philosophy, and a community of adults and kids. This brochure provides you with information about all three of these aspects of Logo.

The Logo programming language is the result of 15 years of research in computer education. It is designed to combine **ease of use**, for the beginning programmer, with **powerful procedural capabilities**, to help the advanced user develop complex applications.

Contents





```
TO SPI :STEP :ANGLE
FD :STEP
RT :ANGLE
SPI :STEP + 2 :ANGLE
END
```

ℾhe Logo Community

ʌe've learned that two computers running
ɹogo in one classroom are much better than
ıne computer in each of two rooms. The rea-
on is that Logo users, even when they're
ɾorking on completely separate projects,
ɩenefit a lot from seeing each other's ideas,
nd helping each other design and debug
ıeir programs.

his sharing of ideas is a vital part of the
ogo style. We'd like to welcome you to a
ommunity of adults and kids, including Logo
evelopers, students, and teachers. This
ɾidespread group shares a culture based on
ɔme ideas about learning and computers,
leas which come from computer science
nd from Piagetian cognitive psychology.

Vho uses Logo?
ogo is best known as a good language for
oung children. Indeed it *is* a good language
ɪr young children, but it is much more than
ıat. It is suitable for high-school students,
ɪr co[illegible]e students, for adults—in fact, for
nyo[illegible]o needs a programming language
ıat is powerful, easy to learn, and easy to use.

ome Education
lany people buy personal computers as
ı educational investment, only to find that
ɔscure programming languages and bad
ɔcumentation discourage them—or their
ıildren—from learning to program.

ɔ encourage the use of Logo in home
ducation, we provide a comprehensive
ːference manual and a special introductory
anual (meant specifically for home learning)
aturing turtle graphics; both books come
part of the Apple Logo package.

hools
ıide from its role in home education,
ɹgo has been used and tested extensively
classrooms from kindergarten to college
vel. See the last page of this brochure for
ɔre information on the use of Logo in schools.

Some Logo Activities

Logo's design reflects an emphasis on learning. You can see this emphasis in the many intriguing application areas built into it.

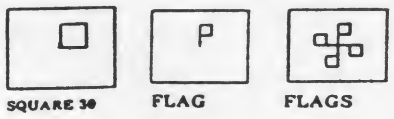
Turtle Graphics
Logo features *turtle graphics*, which makes it possible for beginners to draw beautiful, complex designs the first day. Yet, turtle geometry is based on mathematical ideas powerful enough that it is used in teaching college-level mathematics and physics.

```
TO SQUARE :SIDE
REPEAT 4 [FORWARD :SIDE RIGHT 90]
END

TO FLAG
FD 30
SQUARE 30
END

TO FLAGBACK
FLAG
BK 30
END

TO FLAGS
REPEAT 4 [FLAGBACK RT 90]
END
```



Human Language
Another important Logo application is *human language* processing. A two-line program translates a word into Pig Latin; more complicated programs can generate random grammatical English sentences, carry on simple conversations, or play language games like Hangman.

```
TO LATIN :SENT
IF EMPTYP :SENT [OP []]
OP SE PIG FIRST :SENT LATIN BF :SENT
END

TO PIG :WORD
IF MEMBERP FIRST :WORD [A E I O U] [OP WORD :WORD "AY]
OP PIG WORD BF :WORD FIRST :WORD
END

?PR LATIN [NO PIGS HAVE EVER SPOKEN PIG LATIN AMONG HUMANS]
ONAY IGSPAY AVEHAY EVERAY OKENSPAY IGPAY ATINLAY AMONGAY UMANSHAY
```

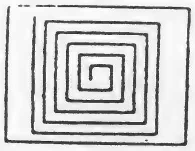
The Logo Philosophy

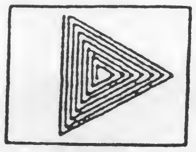
Logo is not just another computer language. It was designed as a tool for carrying out a particular philosophy of education. The goal of this philosophy is to promote independent learning and problem-solving skills through explicit attention to the process of learning itself.

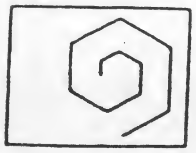
Problem-Solving
For example, consider the problem of getting students to think of learning as a gradual, developmental process, that is, demonstrating a productive view of *partial* understanding. Most students think that understanding comes all at once, or not at all. (They don't often express the idea so explicitly, but they show their belief in conversations which end with "I don't get it.")

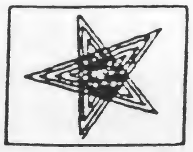
The Logo approach makes it feasible to think about learning styles concretely, by using the computer itself as a model of the learner: in Logo programming, we emphasize the idea of *debugging*. A program which doesn't work isn't simply declared a failure. Instead, it is taken as a partially finished product in need of improvement. This idea is not an artificial one invented just for students: it is a programming commonplace. What's special about the Logo approach to education is that we encourage students *explicitly to use the same idea* in their activities outside of programming.

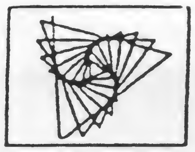
Logo encourages the student to take risks without being threatened by imperfect results.

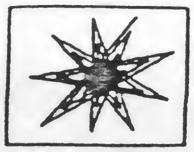

SPI 5 90


SPI 5 120


SPI 5 60


SPI 5 144


SPI 5 125


SPI 5 160

Procedural Thinking
Many other problem-solving ideas are similarly embodied in the Logo programming experience. For example, a well-known technique for attacking a large, complex problem is to break it down into smaller pieces, and solve each part of the problem separately. In Logo, a large programming problem is similarly broken down; the programmer writes a separate *procedure* (a small program which is independent of other procedures, but can communicate with other procedures) for each sub-problem.

Flexibility
Another aspect of the Logo philosophy is its flexibility in curriculum. Logo presents an arena for experimentation. Learning takes place in an exploratory and even playful spirit.

Logo provides tools for exploring areas which many people find immediately appealing, such as turtle graphics. Within these areas, students naturally find challenging projects on their own. A class can be working on ten different projects, rather than on one specific assignment for everyone.

This approach is one reason why Logo is particularly well-suited to individual learning at home, not just to directed group work in a classroom.

There are, of course, many ways to teach the Logo language. If you wanted to do so, you could even teach it in an atmosphere of daily homework assignments and graded weekly quizzes. But by doing so you would forfeit the spirit of flexible exploration that Logo promotes.

Logo learners of any age are doing real research, working on original projects of their own design. There are no right answers for the teacher to dispense. Some of the most important teaching in a Logo classroom is done by fellow students, and a Logo teacher is also a fellow learner.

The Logo Language

Logo is **procedural:** A large Logo program is written as a group of independent parts called procedures. Each procedure has its own variables, and communicates with other procedures in a well-defined way. This characteristic of Logo makes it easy to debug and modify programs, and it encourages a style of problem-solving in which a large problem is broken up into small, manageable bites.

Logo is **interactive:** You can type in a Logo command directly and it will be run immediately. More important, when a program doesn't work completely, you can debug it interactively: run a piece of it, look at the results, change the program a little and try again.

Logo is **extensible:** When you write a Logo procedure, its name becomes a word in the Logo language, just like the built-in procedures. The syntax for using it is the same. Logo can thus be personalized. Teachers and parents can easily tailor the language to their own needs by writing new procedures and teaching them to their kids just as if they were original features of Logo itself.

Logo is **recursive:** A Logo procedure can use itself, as well as other procedures, as instructions within itself. This capability makes Logo tremendously powerful, because certain complex problems include sub-problems which are simpler versions of the original problem.

Logo has **list processing:** A "list" is a data structure for grouping information. The elements of a list can be numbers, letters, words, or other lists, in any combination. Unlike arrays in other languages, a Logo list can change size as a program runs.

Logo is not **typed:** Any Logo variable can take on any type of value. You don't have to know whether the value of a variable will be a number, a word, a list of numbers, a list of lists . . .

Logo's **error messages** are specific and self-explanatory. It has none of the opaque error messages ("SYNTAX ERROR") found in some other languages. For example, if you type FRWARD 50, Logo responds I DON'T KNOW HOW TO FRWARD.

Comparison with BASIC and Pascal

Logo is meant to be both easy for beginners and powerful enough for experienced programmers. "Logo has no threshold and no ceiling." The style of the language allows a smooth growth in the programmer's ability to use its advanced features. Compare this philosophy with that of the two main competing languages for teaching:

	For beginners	For experienced users
BASIC:	**Good interactive style. Small vocabulary.**	Lacks program modularity: no procedures or compound statements.
Pascal:	Hard to get started. Need to use text editor and compiler even for the simplest program.	**Has procedures, compound statements. Advanced data structures can be made using records, with effort.**
Logo:	**Good interactive style. Beginners can start with small vocabulary.**	**Procedures are central to Logo style. List processing makes advanced**

Logo Computer Systems, Inc.
222 Brunswick Blvd.
Pt. Claire, Quebec
Canada H9R 1A6
(514) 694-2885

989 Avenue of the Americas
New York, New York 10018

368 Congress St.
Boston, Ma. 02210

How is Logo Used in Schools?

Logo is used by students of all ages and academic levels. Because it emphasizes non-numeric applications of the computer, Logo programming does not depend on prior mathematical knowledge. Therefore, it can present a fresh chance for students to work with mathematical ideas in a different context.

Logo in the Classroom
Teachers often say to us, "I teach Algebra II (or third grade, or junior high English): how can I use Logo in my classroom?"

One good answer goes something like this: "After you learn Logo yourself, start teaching it to your kids, suggest a few projects to them, and then stand back. With your help, they'll generate a lot of worthwhile ideas on their own. Don't pick one project and assign it to the whole class; give them several ideas, or give each student a suggestion appropriate to him or her."

[illegible] follows is a short list of programming [illegible]ts which are appropriate to various curriculum areas. Bear in mind, though, that one highlight of the Logo philosophy is that it cuts across narrow categories. If you encourage a student who's having trouble with spelling to program a Hangman game, the student will end up thinking about geometry (drawing the scaffold), arithmetic (how many guesses), and problem solving strategies as well as about lists of words.

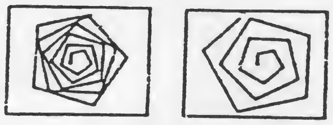
Programming Projects
Exploring polygons with turtle geometry, including the properties of spirals and stars A Logo "polygon" is a much broader concept than the simple closed figure usually studied in geometry. Spirals and stars provide a rich ground for the exploration of geometric concepts that are hard to make accessible to high-school students without the help of Logo.

A *conversational* program, by which the computer carries on an English-like conversation with a human being (on the screen, not out loud). The most famous example of such a program is called DOCTOR, where the computer simulates a non-directive therapist. Writing this kind of program, aside from being a wonderfully challenging problem-solving task, is an excellent way to explore c[illegible]ts of English grammar. It makes gramm[illegible]me alive, by putting it in an environment where it is needed in order to achieve a goal—a conversing program—rather than as an abstract way of analyzing English.

A *French conversation* program (or its equivalent in some other language). This is an excellent project for reasons analogous to those in the previous paragraph. Similarly, try programming a foreign-language version of any game that uses language extensively, perhaps HANGMAN or an Adventure game instead of DOCTOR.

Programming a simulation of natural processes, such as erosion. In connection with, say, an earth science course, students can study what happens in different situations. Although merely using someone else's program in this way can still provide some flexible learning, students who write the program themselves will gain a depth of understanding and a personal involvement that they wouldn't have otherwise.

Using FIRST, BUTFIRST, *and so forth* to explore composition of functions in an algebra class. Abstract functional concepts become easily manipulable in Logo; some students have trouble dealing with functions numerically and find that working with words and lists illuminates the same ideas in a more congenial context.



Recommended Reading

Abelson, Harold. *Apple Logo.* McGraw, Hi (Cambridge, 1981). This book provides a thorough explanation of Apple Logo, along with many detailed suggestions for projects

Abelson, Harold, and Andrea DiSessa. *Turtle Geometry.* MIT Press (Cambridge, 1981). This innovative approach to mathematics uses Logo to let advanced students explore concepts that range from simple geometry to the theory of relativity.

Goldenberg, Paul. *Special Technology for Special Children.* University Park Press (Baltimore, 1979). This book explains the use of computers as a medium of communication for children with severe handicaps.

Papert, Seymour. *Mindstorms.* Basic Books (New York, 1980). This book, subtitled *Children, Computers, and Powerful Ideas;* shares with the reader a vision of education in which children learn powerful ideas through programming computers in Logo.

Apple Logo

Apple Logo, developed by Logo Computer Systems Inc., is available from your local Apple dealer, product D2D0100.

Apple Logo runs on any model Apple II computer with 64K bytes of memory, a moni-

The Friendly Computer Languages

Jim Muller

They're everywhere these days: in homes, offices, laboratories, cars, industrial controls, kitchen appliances, radios and TV sets, security systems, and children's toys. "They" are, of course, the ubiquitous microcomputer, the millions of single chip devices controlling a host of consumer products, the board-level systems controlling industrial processes, and, of course, the desktop computer through which a rapidly increasing number of laypeople are becoming involved in the microcomputer revolution.

To some, this proliferation of computer technology heralds an exciting new revo- n. To others, it is simply revolting.

ecently, however, that negative opinion has begun to change, as new user-friendly languages are introduced for personal computers. No longer does the newcomer to the world of computers have to cope with such alien words as INIT, CHR$, GOSUB, STR$, REM, and DIM. A rapidly growing number of young people and the young-at-heart are entering the world of microcomputers riding on the back of a turtle.

Not long ago, the turtle was an electromechanical pet that roamed the floor, controlled by a sizable computer and a simple child-oriented keyboard. Youngsters could manipulate the turtle through the keyboard to make it draw pictures on large sheets of paper. The turtle had a unique pen that it could raise or lower to draw figures.

In drawing pictures of houses, animals, or just geometric shapes, youngsters gained valuable experience in logical thought and problem-solving. Each task had to be broken down into the simplest steps, and then assembled in a structured procedure to accomplish that task. If the drawing was not correct, they went back a refined the procedure until it was the way they wanted.

Now the turtle resides on the computer display screen in the shape of a small triangle. But it will still draw pictures, responding to simple commands to go FORWARD, BACK, LEFT, or RIGHT. These turtle graphics commands were first introduced to microcomputers through the Logo language, but are now being incorporated into an increasing number of languages and personal computers.

Just as you select a car to suit your own personal tastes, so you should select a turtle graphics language.

Last summer, while watching a group of youngsters manipulate the cybernetic turtle around the screen to create some dazzling graphics, it struck me that these young people needed a place in which they could fully explore the world of microcomputers and microelectronic technology. They needed their own organization through which they could come to know what different computers could do, what languages such as Logo, Pilot, Basic and Pascal were all about. They not only needed the chance to explore the computer, they needed the challenge to excel. From this, the Young Peoples' Logo Association has evolved with a membership of Turtles that span all 50 states, and a growing number of foreign countries.

At first the choices were easy. Throughout 1981, Logo was available only on the TI 99/4 computer. And, through turtle graphics and the sprite mode our young Turtles were soon developing procedures to accomplish all sorts of colorful things. We occasionally used Big Trak, the programmable truck from Milton Bradley, to help demonstrate the Turtle commands of FORWARD, RIGHT, LEFT, and REPEAT to show how a picture had to be broken down into each step the turtle was to take.

By programming Big Trak to go around the Ping Pong table and under the bench, and to fire its phaser at the dog, for example, youngsters were able to visualize the steps needed to guide the Turtles around the screen to accomplish complex geometric tasks.

Atari Pilot

Then came Atari Pilot with turtle graphics, followed quickly by two versions

```
T: "Please enter a distance."      T(ype): "Please enter a distance."
A: #A                              A(ccept): #A (#A is a variable)
GR: *HERE                          GR(aphics): *HERE (Sets up the Label,
                                   "HERE."
GR: PEN RED                        Tells the pen to draw in red "ink."
GR: 4(DRAW #A; TURN 90)            GR(aphics): 4(DRAW #A; TURN 90) Repeat
                                   the operations within the brackets four
                                   times---the turtle will draw a red
                                   line #A units long and then turn right
                                   90o, putting a red box on the screen.
C: #A = #A + 2                     C(ompute): The value of #A = The value
                                   of #A plus 2.
J: *HERE                           J(ump to): *HERE, or to the Label, "
                                   HERE," and repeat the process.
```

Figure 1.

Jim Muller, President, Young Peoples' LOGO Association, Richardson, TX 75081.

of MIT Logo for the Apple II, and then Apple Logo. Turtle graphics is also avail-[illegible] through other languages and systems [illegible] one form or another. Of course, Big Trak has been available for a few years, offering a rudimentary introduction to turtle graphics. This makes it increasingly difficult to recommend what parents and teachers should buy for their own Turtles.

Just as you select a car to suit your own personal tastes, so you should select a turtle graphics language. For example, for the novice driver, the simpler, economy model may be the best vehicle. For under $500, an Atari 400 computer and a colorful, easy-to-use language, Atari Pilot with turtle graphics, is made readily available to youngsters from primary grades up.

In addition to offering enjoyable turtle graphics features and advantages, Atari Pilot offers the unique feature of an easy transition to Basic and other common microcomputer languages. For example, it shares some common format and editing features with Basic.

Pilot, for Programmed Inquiry, Learning, Or Teaching, is a very conversational language which uses simple line abbreviations to prescribe the function of that line. It was developed initially as an easy-[illegible]se authoring system for educators, [illegible] Atari has added turtle graphics features to the language to make it more user-friendly. (See Figure 1.)

If I choose to retain this short program for drawing a box, I can do so by simply entering the command AUTO followed by the label *BOX and the program. To mark the end of the program, enter E: When listed, the program would look like this:

```
10 *BOX
20 T: PLEASE ENTER DISTANCE.
30 A: #A
40 *HERE
50 GR: PEN RED
60 GR: 4(DRAW #A; TURN 90)
70 C: #A = #A + 2
80 J: *HERE
90 E:
```

The program is run by first clearing the screen to set up the graphics mode with GR:CLEAR. Then the Use command is entered "U: *BOX. An endless series of red boxes will be drawn on the screen. However, that can be limited by simply changing line 80 to read:

[illegible](#A 45): *HERE

[illegible], boxes will be drawn until the dimension of a side equals 5.

Boxes and other figures can be placed at any coordinates on the screen. The turtle can draw in yellow and blue, as well as red. There is also a command to FILL a box or a house, so you can make a city of little boxes or colorful skyscrapers.

The language offers sound effects and assembly language access to add some truly spectacular effects. It is also recursive in that commands such as *BOX can be used in other procedures. It also offers some very interesting text manipulation capabilities for easy lesson programming, as well for developing word games.

The greatest asset of the language is spectacular documentation that is heavily illustrated, thorough, and very easy to understand. Also included are two well-done demonstration tapes that orient the user to the capabilities of the language.

However, Atari Pilot has some distinct limitations when compared to the various implementations of Logo. The language is much smaller in size and in scope. But, when it comes to the practical real-world situation of economically providing youngsters with the opportunity to explore the fun and excitement of personal computing, Atari Pilot is an excellent choice and an excellent introduction to the Atari computers. The books that accompany the language do as much to make the computer user-friendly as does the language itself.

The sprite mode is especially useful for elementary grade youngsters.

Three Versions

There are currently three versions of Logo available through four companies: TI Logo from Texas Instruments, MIT Logo for the Apple computer from Terrapin, Inc. and Krell Software Inc., and Apple Logo developed by Logo Computer Systems, Inc. of Canada, and distributed by Apple Computer Inc.

TI Logo requires the basic TI 99/4A console plus Extended Memory and the TI Logo command module. The suggested retail price for the minimum system is just under $1000. The versions of Logo for the Apple II computer require at least one disk drive, extension of the basic memory to 64K, and the language, bringing the suggested retail price for the complete system to around $2400. The language alone is available from Terrapin, Inc. at $149.95 and from Krell Software for $179.95. The Krell version includes an extra demonstration disk called "Alice in Logoland," which takes the user through examples of virtually all of the capabilities of the language. Apple Logo is available for $175 and comes with two excellent, heavily-illustrated books. In terms of documentation, Apple Logo and Atari Pilot are the two very clear-cut standouts.

All of the versions of Logo offer a turtle graphics mode which moves the turtle around the screen to draw geometric shapes and patterns. The Apple versions each offer five pen and background colors whereas TI Logo offers sixteen colors. In the MIT versions of Logo, the shape of the turtle may be redefined and moved around the screen to create animated procedures. TI Logo, however, offers 32 Sprites which may carry or look like 27 shapes. These shapes may be any of five predefined shapes in the language, or they may be defined by the user. The shapes will not, however, draw lines.

Sprites

The sprite mode is especially useful for elementary grade youngsters in that it provides a very colorful means for them to become actively involved with the computer right away. It also provides the means for demonstrating what might otherwise be incomprehensible mathematical concepts.

For example, it is extremely difficult to explain the numerical concept of "zero" to young children. They have no knowledge of what "nothing" is. However, this abstraction becomes very real when they TELL :ALL (of the sprites to) CARRY O. Those sprites that are on the screen will then disappear.

There is, however, an inconsistency in TI Logo that can be quite confusing to young programmers. In the Turtle Mode, it is possible to SETCOLOR [6 15] and have the Turtle draw a red line on a white background. However, in the Sprite Mode, the computer will not accept this as an input. Only the pencolor can be specified with the SETCOLOR command.

A graphics board that provides animation capabilities for Apple Logo was demonstrated at the West Coast Computer Faire in March. This board allows 32 turtles to assume shapes designed into the language or by the user, allows for the use of 16 colors, and sets the turtles in motion with or without the pen drawing a line.

A valuable feature of the versions of Logo for the Apple Computer is the ability to select full screen or partial screen graphics. This can be done even while a procedure is in process. In the Turtle Mode, TI Logo reserves the bottom portion of the screen for text. In the Sprite Mode, the full screen is used with text written from the top of the screen.

Apple Logo allows the user to select the option of having the Turtle "wrap" or not. Simple commands direct the turtle [illegible]erse direction when it comes to the e[illegible]of the screen, or simply to disappear off the screen. In TI Logo, the turtle and the sprite always wrap around.

Logo for the Apple offers considerably more flexibility in both graphic and arithmetic operations through the use of floating point math. This allows for trigonometric, logical, square roots, and other math functions not possible through the TI integer-only system. In addition to the numerical flexibility, the Apple versions of Logo offer more graphic flexibility.

For example, consider the following simple procedure:

```
TO SPIRAL :DISTANCE :ANGLE
FORWARD :DISTANCE
RIGHT: ANGLE
MAKE "D :D + T
SPIRAL :DISTANCE :ANGLE
END
```

Running this simple procedure on the [illegible]e computer will result in the drawing [illegible] ever-increasing spiral. By entering SPIRAL 1 144, a star will be drawn that will increase in size each time it is drawn. The screen will eventually become totally white with only the turtle visible as it continues to draw. In TI Logo, however, the computer will signal OUT OF INK after about 25 repetitions of the procedure.

Apple versions of Logo provide interfacing to printers and other peripherals through assembly language access. The MIT versions of Logo for the Apple offer SAVEPICT and READPICT commands for saving and reading the contents of the screen into and from memory. The screen can also be printed out, a feature that is especially useful for showing screen graphics. Apple Logo does not offer these commands but they can be programmed. TI Logo offers no interfacing capability. Procedures can be printed on the TI Thermal Printer, and programs can be stored on tape or disk.

Logo and the Handicapped

Imagine the use of Logo by the physically handicapped wherein they have access to single-key commands that print [illegible]hrases and sentences. People with [illegible] arthritis or other problems that limit their ability to type could still use the computer to communicate by entering a single key command.

For example:

```
TO G
CLEARSCREEN
 PRINTER
PRINT [YOUNG PEOPLES' LOGO ASSOCIATION]
PRINT [1208 HILLSDALE DRIVE]
PRINT [RICHARDSON, TEXAS 75081]
PRINT [ ]
PRINT [DEAR YPLA:]
END:
```

By simply typing the letter G and enter, the address of the YPLA and opening of the letter would be printed out on the printer. With a menu of useful phrases and sentences, the handicapped have a new means of communication readily available to them. And this list can be altered very easily to serve new requirements.

Another benefit of assembly language access through Apple versions of Logo is the ability to create new features in the language, such as music and sound effects. This also gives the user access to

The real power of Logo for the Apple computer becomes evident with the list processing capabilities offered by each.

16-color low-resolution graphics and the chance to alter the character set. TI Logo allows direct access to the character set by which youngsters have created their own alphabets and codes. There is no access to music or sound capability in this version, however.

The value of TI Logo is in its graphic capabilities and its user-friendliness for very young children. When the idea for the YPLA was being formulated last summer, junior high students and first graders used to get together around a few computers on a Ping-Pong table with the intent of duplicating their favorite arcade games through the computer or simply drawing colorful flowers. Not only did they all find the TI 99/4 exceptionally easy to use, they found that TI Logo offers very easy access to some exciting graphic and game programming capabilities.

We have had a variety of games developed including versions of lunar lander, flight control, crossing the asteroid belt, and similar exercises. Primary grade youngsters have developed some very interesting pictures, not to mention our first computerized Christmas card. The "human factors" designed into the language make it an excellent choice for elementary and junior high schools, and for those age groups at home. Unfortunately, the documentation that accompanies the language leaves a lot to be desired.

For example, a variety of undocumented commands have been found in TI Logo. FPUT, LPUT, THING, NUMBER?, WORD?, THING? are among these. The manual accompanying TI Logo was supposedly written for middle school age children, which makes it difficult to understand why the joystick commands, JOY1 and JOY2, were omitted.

Not only do these commands make the language competitive with several other child-oriented software packages, they offer some very interesting opportunities for using TI Logo with the handicapped. Each position of the joystick represents a different numerical input, and a simple rewiring or rebuilding of the joystick would offer the physically handicapped access to the computer.

Documentation

The documentation problems with TI Logo are reportedly being corrected, and a TI Logo Curriculum Guide is being published. In addition, several authors are preparing books on TI Logo for various age groups. Thus, this problem should disappear, hopefully as this article is being published.

The real power of Logo for the Apple computer becomes evident with the list processing capabilities offered by each. These capabilities tend to make these versions more complete computer languages. However, there are some differences worth noting. The MIT versions offer some editing and debugging commands which have to be programmed into Apple Logo. On the other hand, error handling in Apple Logo is generally more efficient in that error handling commands can be defined by the user.

The MIT versions of Logo for the Apple and the TI 99/4 were experimental developments. It would appear that Logo Computer Systems, Inc. learned from these experiments and has come up with a more complete language. Of course, it is reasonable to assume that all of the Logo developers are looking at improvements in the language. Thus, it will take more time to determine which version is going to become the "standard."

For the parent and teacher, there are, indeed, some tough choices. For the economy-minded, Atari Pilot is an excellent choice, the subcompact model that

MERRY BEE

[illegible]	Ages 5-6	$29.95
Word Games	8 to Adult	$24.95
Early Words	Ages 5-7	$19.95
Notes	Music Drill	$19.95

Each disk includes 3-12 separate programs.

For fun and educational too.
Full use of graphics and sound.
Apple II+ 48K. Specify 3.2 or 3.3.
Add $2.00 shipping per order.

MERRY BEE COMMUNICATIONS
815 CREST DRIVE
OMAHA, NE 68046
(402) 592-3479

Specializing in Apple software.
Consultants in interactive video.

Ap[illegible] is a trademark of Apple Computer Co.

CIR[illegible] 201 ON READER SERVICE CARD

NEW Kits Now Available

NEWPORT "Pro-Sticks"
the Mercedes of Joysticks
'Creative Computing Magazine' Aug. 1982

Soft-Touch Controllers
for busy fingers and fast action

NEW Extension "Y" Adapters
with rapid fire action

Build your own custom video-game controls from our proven high quality arcade components.

Create your own control console right in your home. Send $3.00 for a brochure of creative and innovative ideas in custom video game joysticks and controllers. Includes easy to follow schematics and instructions for Atari, Apple, Pet Commodor, TSR-80, and other computers, and a complete kit and components price list.

INQUIRE ABOUT OUR CLUB DISCOUNTS

Please rush me ____ brochures at $3.00 each (includes First Class Mail and handling) and mail to:

Name ____________

Address ____________

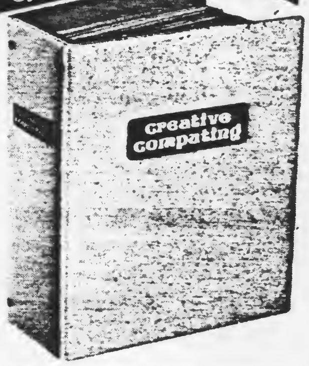
City ________ State ____ Zip ____

NEWPORT MACHINE DESIGN
P. O. Box 418, Bishop, CA 93514
(714) 873-5677

CIRCLE 227 ON READER SERVICE CARD

creative computing

BINDERS



Are your back issues of *Creative Computing* stacked in piles? In boxes? Does the one you want always seem to be missing? If so, you need our binders. Made of vinyl with the *Creative Computing* logo silk screened on front and spine. Each holds 12 issues with sturdy locking wire posts. Just $7.95 each. Or save by ordering 3 for $22.50, 6 for $42.95.

CHARGE ORDERS PHONE 24 HOURS
TOLL FREE 800-631-8112
In N.J. only (201) 540-0445

creative computing Dept. C924, 39 E. Hanover Avenue, Morris Plains, NJ 07950

Please send me ____________ *Creative Computing* Binders.

☐ PAYMENT ENCLOSED $ ____________ (NJ residents add 5% sales tax.)
Add $1.00 per order for postage and handling. Outside USA add $2.50 per unit ordered. Send U.S. funds only.

☐ CHARGE MY:
☐ American Express ☐ MasterCard ☐ Visa

Card Number ____________ Exp. Date ____________

Signature ____________

Mr./Mrs./Ms. ____________
(please print full name)

Address ____________

City/State/Zip ____________

Friendly Computer, continued...

offers excellent economy combined with efficient performance. This is not only because of the user-friendly benefits but because of its similarities to other more readily available programming languages. TI Logo is an expensive version of Logo, the intermediate model offering some unique benefits that will undoubtedly be improved and expanded in time. The TI computer is gaining rapidly in popularity and is being more actively supported by third parties, a good indication of the viability of the product.

The top-of-the-line resides within the Apple computer. To carry the analogy a step further, consider the version of Logo offered by Terrapin, Inc. as the solid, comfortable, efficient, four-door sedan. It is an excellent, long term investment backed by an experienced company. But would you teach a first time driver in a Mercedes or Cadillac El Dorado?

The Krell Software version appears to have more of a "luxury sports car" approach. It is a lean, dramatic package that offers comprehensive and colorful demonstration procedures. But in total, it is the model that requires some prior experience to use most efficiently.

Apple Logo is the Grand Touring model—solid, comfortable, dramatic, luxurious, well-documented, with outstanding performance. It stands out because of its thorough attention to detail.

In our Turtle Learning Centers in Irving and Richardson, TX, we use them all. Each has its own followers and supporters. Each language has its own advantages. But, most important, there is something for everyone.

Many computer owners may find it hard to relate to that statement. This is why the analogy of languages to automobiles goes far beyond selection in a showroom. A few years ago, I was actively involved in promoting an employer's products to the road racing field. Race drivers, the hobbyists of the auto industry, were no more fiercely loyal to their favorite manufacturers than are computer owners.

It is from the development work of the hobbyists that many consumer products have evolved—cars, radios, TV, hi-fi, stereo, and now the computer. Hopefully, the new friendly languages will be the development to make consumer products out of computers. ☐

MicRo Information Systems
a division of M-R Information Systems, Inc.
Presents Educational Software designed to motivate the student.
(programs available for the TRS-80 Mod III* and VIC)**
Write for complete catalog, to:
M-R Information Systems, Inc.,
P.O. Box #73, Wayne, N.J., 07470
* TM Tandy, ** TM Commodore

CIRCLE 220 ON READER SERVICE CARD

Logo—A Cultural Glossary

E. Paul Goldenberg
Lincoln-Sudbury Regional High School
390 Lincoln Rd.
Sudbury, MA 01776

For easy access, the terms in this glossary are arranged alphabetically, but this arrangement hides the complex interrelationships of the ideas in the definitions. The groupings on this page were developed to make obvious the relationships between important Logo concepts. You can use it as a map to guide you through the glossary.

The groupings do not represent a true tree structure because some of the terms appear under several different headings. The main concepts are organized into broad categories, with more detailed information listed in outline form under each main heading.

The Logo Language

- program
  - procedures
    - command
    - operation
    - primitive
    - subprocedure
  - workspace
    - global
    - local
    - state
  - recursion
  - interation
  - input and output
  - object
    - word
    - list
      - sentence
  - turtle
    - turtle graphics
      - turtle geometry
        - Total Turtle Trip Theorem
        - playing turtle (body geometry)
      - forward (back)
      - left (right)
    - dynaturtle
    - sprite

Seeds of the Logo Culture

- artificial intelligence (AI)
  - microworlds
  - LISP
- Piaget
  - debugging
- QWERTY phenomenon

Learning and Thinking

- interest worlds
- microworlds
- powerful ideas
  - state
  - recursion
  - subproceduralization
  - metaphors
    - prosthetic
    - anthropomorphic images
    - playing turtle
    - teaching
- problem solving
  - debugging
- controlling the computer
  - computer literacy
  - teaching
    - programming

Outside the Logo Culture

- computer-aided instruction (CAI)
- computer literacy

This glossary is more cultural than technical partly because the learning philosophy behind Logo is more of a culture than a technique. The Logo technical vocabulary consists, for the most part, of familiar words adopted from general usage to refer to specific Logo metaphors, images, and ideas. Many of the terms and definitions presented here are derived from three books: Seymour Papert's *Mindstorms: Children, Computers, & Powerful Ideas* (New York: Basic Books, 1980); Harold Abelson and Andrea diSessa's *Turtle Geometry: The Computer as a Medium for Exploring Mathematics* (Cambridge, MA: MIT Press, 1981); and my own *Special Technology for Special Children* (Baltimore: University Park Press, 1979). Reference is also made to Brian Harvey's article "Why Logo?" on page 163 in this issue of BYTE.

anthropomorphic images: metaphors in which computers, computer procedures, and objects controlled by computers are thought of as if they were persons. "Anthropomorphic images facilitate the transfer of knowledge from familiar settings to new contexts. For example, the metaphor for what is usually called programming computers' is teaching the turtle a new word." (See Papert, page 59.) Thinking of the machines as (limited) people—and even modeling people's behavior through analogies to machine processes—does not involve treating people like machines. (On this latter point, see the entry on **computer-aided instruction**.)

About the Author
While working with the Logo Group at MIT, E. Paul Goldenberg pioneered in applications of Logo for education and rehabilitation of children with severe communication handicaps. His book Special Technology for Special Children *presents the philosophy, psychology, and technology behind this work and illustrates it with case histories. He is currently Assistant Professor of Rehabilitation Medicine at Tufts Medical School and will soon be director of the computer department at Lincoln-Sudbury Regional High School.*

artificial intelligence (AI): the branch of computer science from which Logo grew. The name of this science derives from its attempts to simulate, using machines, the behavior that is regarded as intelligent in people or animals. If this narrow view of the field was ever true, it no longer is. Cognitive psychology and artificial intelligence together are sometimes referred to as cognitive science. They study such disparate processes as natural-language understanding, visual perception, and knowledge acquisition. Good studies of human information processing frequently require both a careful study of how people perform tasks and serious attempts to build models based on a theory and observations to test the theory. The complexity of these models requires computer simulation.

command: a Logo procedure that performs in some particular way, but does not return a value to its calling procedure. It's analogous to a procedure in Pascal. See also **operation**.

computer-aided instruction (CAI): in the broadest sense, any educational endeavor that is aided by computers. In general, however, CAI means automated worksheets (drill and practice) or electronic tutors (frame-oriented CAI or automated programmed texts). This is very different from the Logo philosophy of using the computer not as the supplier or exerciser/tester of knowledge, but rather as a context within which to use thinking to solve problems of genuine interest.

Programmed learning explicitly models the human as a machine in that the student is being programmed by the computer. Logo learning sees the learner as the agent—actively constructing knowledge. The student takes the teacher's role—teaching the turtle a new word.

computer literacy: often seen as a general (and superficial) experience with computers. This concept is tied to a transitional stage in the spreading of computer technology. When computers were rarer and more specialized, computer literacy was not an issue (just as no one now worries about electron-microscope literacy). When computers become as common as cars, computer literacy will cease to be an issue.

Literacy has two conventional meanings. The first—being well-read and articulate in one's language—suggests fluency in a particular computer language. Programming is having the ability to express a novel idea in that language. Letting children be programmers, helping them to become fluent at expressing mathematical and logical ideas, is the Logo sense of a thoughtful literacy. (This idea, however, doesn't fit in with the image that the word "literacy" is generally used to convey, a skill that every child must learn in order to be able to cope. It's more like learning a foreign language: a valuable skill for those who choose to learn it, but not an absolute requirement for life or for employability.)

The other conventional meaning of literacy—having minimal reading skills—makes little sense in a society replete with computers. With many computer languages to choose from, what could be considered minimal communication skills? Teaching general familiarity with computers without providing an opportunity to develop good communication skill is a little like having a course teaching people the names of the features of a car without allowing them to learn to drive.

controlling the computer: the issue is one of locus of control. Whereas computers have conventionally been used in education to program the kids—in effect, to control their behavior—the Logo philosophy stresses kids programming computers. It is often said that teaching is the best way to learn. The computer is a highly responsive student and rigidly faithful to its teacher. For students who have had little sense of control in school, this, even apart from the content of the subject they are studying, is a valuable experience. (The importance of being in control is particularly apparent in the Logo work of students with special needs. For more, see *Special Technology for Special Children*.

debugging: improving the behavior of a program that does not do what you want it to. Logo emphasizes that programs that do not work as desired are merely unfinished, rather than bad finished products. This is in sharp contrast to many school situations in which a task is considered to be over when a particular time has come, rather than when it "works."

In school, a written composition tends to be graded as it stands, rather than debugged. (If the teacher does encourage a student to improve a paper, the effort involved can be prohibitive.) Logo teachers encourage students to love their bugs. If a turtle does not do what you wanted it to, chances are that it does something interesting anyway. The unwanted behavior may lead to new ideas more interesting than the original task. A bug can be neat! It can certainly teach you something . . . study it!

dynaturtle: a dynamic (rather than static) turtle. Whereas a static turtle has a fixed spatial position and heading, a dynaturtle may have a fixed velocity or acceleration. Whereas commands (state-change operators) to a static turtle specify a change in position (e.g., FORWARD 100) or heading (e.g., LEFT 90), a command to a velocity turtle specifies a change in velocity and has an analogy to force. Dynamic turtles can be a flexible laboratory tool for experimentation in physics and mathematics and, like their static cousins, for aesthetics as well.

FORWARD: Logo command to the turtle telling it to move forward in the direction it is heading. This state-change operator takes a single input that says how far (in turtle-steps) forward to move. For example, FORWARD 100 says move forward 100 units; FORWARD :SIDE + 10 says move forward the distance 10 + :SIDE (whatever value the variable :SIDE happens to have at the moment). The only other turtle-geometric state-change operator dealing with the turtle's position is called BACK. Neither of these change the turtle's heading. See also **LEFT**.

global: something pertaining to the entire environment being considered. Among variables, global variables are those that can be accessed and changed from any place in a program. Logo encourages the use of local variables—variables that belong to a particular procedure—because they make for more orderly and more debuggable programs.

Another kind of global reference is the Cartesian coordinate system. It is global in that each point is specified in relation to a standard referent (the origin). Points are not specified in relation to each other. For a discussion of the difficulties caused by such a global perspective for graphics representation, see the section of Brian Harvey's article "Why Logo?" on page 178 in this issue of BYTE. Logo's orientation is, in general, toward local references. See **local**.

input and output: within the Logo culture, we pay more attention to the inputs and outputs of procedures than to traditional issues of hardware. The input to a procedure is thought of as a message that the procedure needs in order to do its job. As an example, FORWARD needs an input telling it how far to move the turtle; PRINT needs to know what to print. Some procedures need more than one input message in order to know what to do. A procedure to draw arbitrary polygons, for example, needs two inputs, one telling it the size of its forward step, and the other telling it the angle to turn at each corner.

The output of a procedure is a message sent back to the procedure that called it. For example, SUM outputs a message that is the sum of its two inputs. A procedure that produces output (see **operation**) must send its message to a procedure expecting an input. (In this way, output and input are closely linked in Logo.) Hardware aspects of I/O are handled conveniently for the programmer so she can concentrate on the behavior of the conceptual building blocks (procedures) she is creating without wasting attention on the machine.

interest worlds: areas of special interest in which the computer can be a useful tool, servant, or laboratory. Art, music, geometry, physics, and language have all been extensively developed as interest worlds by various Logo investigators. See also **microworlds**.

iteration: telling the computer to execute something repeatedly. See **recursion**, which Logo favors as a control structure.

learning: the focus of the Logo environment. What one does for oneself. See **teaching**.

LEFT: Logo command to the turtle telling it to turn left while remaining in the same location. This state-change operator takes a single input that says how many degrees to turn. For example, LEFT :ANG says turn left :ANG degrees (whatever value the variable :ANG happens to have at the moment). The only other turtle-geometric state-change operator dealing with the turtle's heading is RIGHT. See also **FORWARD**.

LISP: acronym for LISt Processing. A programming language widely used in artificial-intelligence research, the basis for many of the ideas in Logo.

list: Logo's fundamental data structure. A list is an ordered sequence of arbitrary Logo objects (see **object**). Since its elements may be either words or other lists (which may themselves contain yet other lists, nested to any level), lists can be used to create very complex data structures. They can represent information trees (decision trees, binary trees, etc.) and unordered sets. Lists are sometimes used in Logo to accomplish the same purposes for which arrays might be used in other languages. (Logo also has other ways of providing access to information that is indexed numerically or with alphabetic labels. Some Logo implementations have array-handling primitives.) The simplest elements of lists are words.

local: something that has meaning only in the specific context in which it is being used. A local variable is one whose name has a value only within the procedure in which it is defined. Most variables used in Logo are local. The turtle's perspective on its turtle-moves is also a strictly local one. It does not know where it is (with respect, say, to a globally defined origin). In the grand scheme of things, it doesn't need to know where it is. It only needs to know how to move with respect to itself. "The turtle can forget about the rest of the plane when drawing a circle and deal only with the small part of the plane that surrounds its current position.

"By contrast, $x^2 + y^2 = r^2$ relies on a large-scale, global coordinate system to define its properties. And defining a circle to be the set of points equidistant from some fixed point is just as global. . . . The turtle representation does not need to make reference to that 'faraway' special point, the center." (See Abelson and diSessa, page 14.) This local view of movement in space is not only easier to use for simple mathematical ideas, but lends itself quite beautifully to extensions into very fancy math; calculus and limits immediately come to mind. See **global.**

Logo: not an acronym. Derived from the Greek word for "word" or "thought." The name was coined by Wallace Feurzeig at Bolt Beranek and Newman Inc., one of the collaborators in the development of the language.

metaphor: Logo learning makes considerable use of metaphor and pays particular attention to which metaphors are chosen. Is a computer a tutor, a student, a pair of eyeglasses, or a screwdriver? Is a variable name the name by which we refer to a particular value, or the name of a box in which we find whatever we find? Are procedures little folks with specific jobs to do? Are robot turtles literate but literal-minded pets? It is not that Logo learners have particular metaphors that are different from those of others—some people find the little-people model of procedures useful, while others find it irritating—but that the use of metaphor is such a natural part of Logo learning.

microworlds: as used by Logophiles, a microworld is a well-defined, but limited, learning environment in which interesting things happen and in which there are important ideas to be learned. A microworld can have other microworlds within it. For example, within the microworld of turtle graphics, one can define a smaller microworld consisting of all the designs that can be drawn with a POLYSPI procedure. (See R. W. Lawler's "Designing Computer-Based Microworlds" on page 138 in this issue.)

The concept of microworlds is borrowed from artificial-intelligence research. It's very difficult to simulate intelligent behavior in general, but by restricting our attention to a very small area we can begin to find elements that can be modeled. Thus, the concept of microworlds is a useful source of interesting programming projects.

The most famous AI microworld is the blocks world, in which the computer controls a robot arm that can manipulate small blocks. You can tell the computer things like "Put the red block on top of the small green block," or ask questions like "Are any green blocks under a red block?" This is a microworld of the English language (among other things) in that it must understand the vocabulary, syntax, and semantics of sentences pertaining to the moving of blocks. It is also a microworld of structural stability, in that it must understand what physical maneuvers are possible with stackable blocks. One cannot, for example, realistically pick up the bottom block in a stack of eight without dropping others. Similarly, one can place a pyramidal block on a

INVESTMENT ANALYSIS FROM CENTENNIAL

CENTENNIAL SOFTWARE / 410 17TH ST. SUITE 1375 / DENVER, CO 80202 / (303) 595-9193

STOCK-FOCUS

Find out how low is low and how high is high. Using capital structure and performance data, Stock-focus objectively calculates the underlying value of a stock. The system was first developed by the management science department of a major money center bank, and is now in use by investment advisors, trust companies and brokerage houses. On your screen, Stock-focus will plot an estimate of lowest value, highest value and the current price. You then decide what to buy, sell or hold.

REAL-FOCUS

Exhaustively analyze potential real estate investments using the Wharton School's approach to real estate analysis. In minutes you can project profit, costs, and IRR for any project over a 10 year period. Real-focus accounts for amortization, debt, income, operating expenses, taxes, depreciation, and cash flows for both after-tax holding and the results of sale. With Real-focus you can analyze any potential investment from a single building project to a complex time-phased planned unit development.

THE FOCUS TECHNIQUE

FOCUS is CENTENNIAL SOFTWARE's new approach to Program Architecture, providing a natural interaction between microcomputers and users. It provides worksheet style input screens, free access to all program segments, and the ability to combine individual results files for portfolio analysis. You also receive a usable reference manual, menu helps, a program glossary, multiple report formats and spooling. With our FOCUS technique even the most complex programs are versatile and easy to use.

COMPUTER	MEMORY-K	REAL-FOCUS	STOCK-FOCUS
IBM PC	64	$179.00	$189.00
Apple (plus) II	32/48	149.00	159.00
TRS-80 I	32/48	149.00	159.00
TRS-80 II	64	179.00	189.00
TRS-80 III	32/48	149.00	159.00

ALL PROGRAMS IN DISK BASIC

IBM is a trademark of IBM. Apple is a trademark of Apple Computer Corp., TRS-80 is a trademark of Tandy Corporation.

ORDER NOW! FILL IN OR CALL 800-525-2003 (Toll Free)

PROGRAM NAME ________ COMPUTER ________ MEMORY ________

NAME ________ PHONE # ________

ADDRESS ________ ☐ ☐

____

ACCOUNT # ________ EXP. DATE ________

Circle 282 on inquiry card.

FOR TRS-80 MODEL I OR III
IBM PERSONAL COMPUTER

- **MORE SPEED**
  10-20 times faster than interpreted BASIC.
- ★ **MORE ROOM**
  Very compact compiled code plus VIRTUAL MEMORY makes your RAM act larger. Variable number of block buffers. 31-char.-unique wordnames use only 4 bytes in header!
- ★ **MORE INSTRUCTIONS**
  Add YOUR commands to its 79-STANDARD-plus instruction set!
  Far more complete than most Forths: single & double precision, arrays, string-handling, clock, graphics, (IBM low-res. gives 16-color or 200-tint display).
- ★ **MORE EASE**
  Excellent full-screen Editor, structured & modular programming
  Word search utility
  THE NOTEPAD letter writer
  Optimized for your TRS-80 or IBM with keyboard repeats, upper/lower case display driver, full ASCII.
- ★ **MORE POWER**
  Forth operating system
  Concurrent Interpreter AND compiler
  VIRTUAL I/O for video and printer, disk and tape
  (10-Megabyte hard disk available)
  Full 8080 or 8086 Assembler aboard
  (Z80 Assembler also available for TRS-80)
  Intermix 35- to 80-track disk drives
  IBM can read, write and run M.3 Disks
  M.3 disks can read, write and run M.1 disks

mmsFORTH

THE PROFESSIONAL FORTH SYSTEM
FOR TRS-80 & IBM PC

(Thousands of systems in use)

MMSFORTH Disk System (requires 1 disk drive, 32K RAM)
V2.0 Radio Shack TRS-80 Model I or III $129.95*
V2.1 IBM Personal Computer, (80-col. screen) $249.95*

AND MMS GIVES IT PROFESSIONAL SUPPORT

Source code provided
MMSFORTH Newsletter
Many demo programs aboard
MMSFORTH User Groups
Inexpensive upgrades to latest version
Programming staff can provide advice, modifications and custom programs, to fit YOUR needs.

MMSFORTH UTILITIES DISKETTE: includes FLOATING POINT MATH (BASIC ROM routines plus Complex numbers, Rectangular-Polar coordinate conversions, Degrees mode, more), (TRS-80) a full Forth-style Z80 ASSEMBLER) plus a powerful CROSS-REFERENCER to list Forth words by block and line; plus all on one diskette (requires MMSFORTH V2.0, 1 drive & 32K RAM) $39.95*

FORTHCOM: communications package provides RS-232 driver, dumb terminal mode, transfer of FORTH blocks, and host mode to operate a remote TRS-80 (requires MMSFORTH V2.0, 1 drive & 32K RAM) $39.95*

THE DATAHANDLER: a very sophisticated database management system operable by non-programmers (requires MMSFORTH V2.0, 1 drive & 32K RAM) $59.95*

FORTHWRITE: fast, powerful Word Processor w/easy keystrokes, Help screens, manual & demo files. Full proportional w/tabs, outdenting. Include other blocks, documents & keyboard inputs— ideal for form letters (requires MMSFORTH V2.0, 2 drives & 48K RAM) $175.00*

MMSFORTH GAMES DISKETTE: real-time graphics & board games w/source code. Includes BREAKFORTH, CRASH-FORTH, CRYPTOQUOTE, FREEWAY (TRS-80), OTHELLO & TICTACFORTH (requires MMSFORTH V2.0, 1 drive & 32K RAM) $39.95*

Other MMSFORTH products under development

FORTH BOOKS AVAILABLE

MMSFORTH USERS MANUAL - w/o Appendices $17.50*
STARTING FORTH - best! $15.95*
THREADED INTERPRETIVE LANGUAGES - advanced, analysis of FORTH interals $18.95*
PROGRAM DESIGN & CONSTRUCTION - intro. to structured programming, good for Forth $13.95*
FORTH -79 STANDARD MANUAL - official reference to 79-STANDARD word set, etc $13.95*
FORTH SPECIAL ISSUE, BYTE Magazine (Aug. 1980) - A collector's item for Forth users and beginners $4.00*

• ORDERING INFORMATION: Software prices include manuals and require signing of a single computer license for one-person support. Describe your Hardware. Add $2.00 S/H plus $3.00 per MMSFORTH and $1.00 per additional book: Mass. orders add 5% tax. Foreign orders add 20%. UPS COD, VISA & M/C accepted; no unpaid purchase orders, please.

Send SASE for free MMSFORTH information
Good dealers sought

Get MMSFORTH products from your computer dealer or

MILLER MICROCOMPUTER SERVICES (B8)

cube, but not a cube on a pointy block.

object: the naturalness with which Logo passes data messages back and forth among procedures (see **input and output**), and the ubiquity of anthropomorphic images for those procedures, makes the metaphor of object manipulation very appealing for what tends to be called data processing elsewhere. Logo has two kinds of data objects, words and lists. (Some implementations also provide arrays.) Though procedures are generally known as the active elements in a workspace, they can also be manipulated (created, edited, destroyed, or passed back and forth) by other procedures. Therefore, Logo's procedures are also frequently referred to as objects, especially when one is referring to the contents of the workspace. Finally, "object" retains all of its conventional meanings in addition to the technical sense mentioned already. Thus, we speak of procedures as manipulating objects, whether the objects are words printed on the screen or turtles doing a dance on the floor.

operation: a Logo procedure that computes a value and returns that value to the calling procedure. (It is analogous to what Pascal calls a function.) The value can be any Logo object—a number, word, or list. See also **command**.

Piaget: Jean Piaget, one of the great thinkers of our time, realized the value of looking at a wrong answer and trying to understand it (see **debugging**). He was a keen observer of children and recognized that every learner (in particular, each infant and child) takes an active role in his or her own development. For each wrong answer a child came up with (e.g., the 4-year-old stating confidently that trees make the wind blow), he asked the question: Why *that particular* wrong answer? What is the *logic* in the child's thinking that leads to that kind of explanation?

In this sense, he saw these wrong answers not as random movements in [illegible] of the right answer, but as the result of bugs in a program that does give mostly the right answer. The frequent focus of attention on Piaget's stages is a bit of a red herring. Although he has described these stages discretely, it is their contents—the form that thinking takes at various developmental levels, the *logic* of it—and not the stages themselves, or the age at which they appear, that are the real importance of Piaget's theory.

Evidence of the influence of Piagetian thinking (if not each of Piaget's specific notions about thinking) pervades the Logo culture. Our interest in debugging, the metaphor of objects, and the assimilation of computer technical ideas into familiar contexts (e.g., anthropomorphization or playing turtle) all reflect that influence.

playing turtle: pretending to be the turtle and walking through a turtle-graphics procedure as the turtle might see it. This process can make fairly difficult geometric constructions transparent to young children with little or no formal training in geometry. Playing turtle, though, refers as much to the thinking through of a procedure before programming it as to walking through an existing procedure.

By way of example, some 10-year-olds were trying to figure out how to teach the turtle to make a circle. I suggested they play turtle. They concluded that if all they could tell the turtle was to go FORWARD and to turn, it would have to go just a little bit forward, turn a little, go a little bit forward again, turn a little again, and so on. After one kid had made it around the circle (one of the instructions was "keep doing that until you get back to the beginning"), they were convinced they had the right idea. To help them with the details, I reminded them of the Total Turtle Trip Theorem and again encouraged them to play turtle. They began to reason out the details of how much to turn, how many times to repeat the process, and how big they wanted to make each step.

powerful ideas: Seymour Papert compares the Total Turtle Trip Theorem to "its Euclidean counterpart: (at least in the context of Logo computers), the Total Turtle Trip Theorem is more powerful: The child can actually use it. Second, it is more general: It applies to squares and curves as well as to triangles. Third, it is more intelligible: Its proof is easy to grasp. And it is more personal: You can 'walk it through,' and it is a model for the general habit of relating mathematics to personal knowledge." (See Papert, page 76.) An idea that can be used in a variety of personally meaningful contexts, that can be thought through, and that is a model for clear thinking is certainly powerful. Other powerful ideas are state, recursion, and subproceduralization.

primitives: the built-in procedures of Logo. Since Logo encourages programmers to build their own procedures as chunks of larger projects, it is useful to have a term to refer to the indivisible chunks that Logo provides initially.

problem solving: a skill that is at the core of Logo's "curriculum." Most traditional math instruction is about already-solved problems; the student memorizes someone else's techniques. Only recently have people understood that real mathematicians do something very different; they don't just study old problems, they solve new ones. Professor George Polya at Stanford University has been a pioneer in studying the techniques that good mathematicians bring to bear on new problems. Many ideas that are part of the Logo programming style parallel Polya's mathematical ideas. Most important, the use of procedures as building blocks for more complex procedures (as opposed to writing one huge program without a layered structure) parallels Polya's strategy of dividing a large problem into smaller pieces.

procedures: the conceptual building blocks of Logo programs. Logo users start with a vocabulary of primitive (built-in) procedures, and use them to teach Logo new vocabulary that can then be used in all the ways that primitives are used.

program: the Logo use of "program" is much more like the television use of it than of the usual computer use. A program is the whole show and may consist of a host of songs and dances, skits, acts, and routines. (But the word "routine" is not routinely used by Logophiles to mean a part of a computer program.) A person writes procedures. The top-level procedure runs a program that may be quite simple or may involve the use of several subprocedures. If it makes use of subprocedures, this top-level procedure may be referred to as a superprocedure.

programming: consider the following metaphors for programming: teaching turtle a new word; communicating with the turtle; translating from English to French or Logo; or being fluent enough in a language to express oneself easily.

prosthetic: one metaphor for the computer. As an artificial arm may help an amputee manipulate objects, so may an artificial piano player help one manipulate music. Because it is a versatile tool it can provide access to a large variety of inaccessible spaces and activities. The computer's use in musical composition and performance, graphics production (e.g., the special effects in a slew of recent television advertisements and in movies like *Star Wars*), and other areas are all extensions of human abilities—but extensions without which we are severely disabled in areas we now take for granted. It is merely a sociological and technological artifact that we regard as handicapped people who lack certain other abilities we take for granted. (For more on computers as aids to communication and autonomy for special education or rehabilitation, see *Special Technology for Special Children.*)

QWERTY phenomenon: Papert uses this term to refer to traditions that dig

BYTEWRITER

DAISY WHEEL PRINTER

LETTER QUALITY PRINTER AND TYPEWRITER IN ONE PACKAGE

The BYTEWRITER is a new Olivetti Praxis 30 electronic typewriter with a micro-processor controlled driver added internally.

$795 plus shipping

Dealer Inquiries Invited

FEATURES
• Underlining • 10, 12, or 15 characters per inch switch selectable • 2nd keyboard with foreign grammar symbols switch selectable • Changeable type daisy wheel • Centronics-compatible parallel input operates with TRS-80, Apple, Osborne, IBM and others • Cartridge ribbon • Typewriter operation with nothing to disconnect • Service from any Olivetti dealer • Self test program built in.

BYTEWRITER

125 NORTHVIEW RD., ITHACA, N.Y. 14850
(607) 272-1132

Praxis 30 is a trademark of Olivetti Corp.
TRS-80 is a trademark of Tandy Corp
BYTEWRITER is a trademark of Williams Laboratories

themselves in after their original purpose, presumably a good one, has become obsolete. The name derives from the six top left-hand letters on a standard typewriter keyboard. This arrangement of keys was chosen to slow down a typist so that the early manual typewriters would not readily jam. Attempts to convert to a more optimal arrangement of keys have routinely failed as people were used to doing things the old way. So, too, much of the curriculum survives primarily because it is familiar and not because it makes much current sense.

recursion: a recursive definition defines a procedure or function in terms of itself. For example, a recursive definition of "factorial" states that 0!=1 and that n! = n*(n−1)! (See **recursion.**) A Logo program based on this definition might look like this:

```
TO FACTORIAL :N
IF :N = 0 OUTPUT 1
OUTPUT :N * FACTORIAL (:N - 1)
```

A nonrecursive definition might say something like:

```
n! = n*(n-1)*(n-2)*(n-3)*
        ...*3*2*1.
```

(Compare the structures by writing the latter in BASIC.) Both definitions and programs work, of course, but which definition one chooses affects how one programs. In the case of "factorial," both definitions are easy to implement, but sometimes recursion is the only way. For more on recursion, see the section of Brian Harvey's article on page 166 in this issue of BYTE.

sentence: in Logo, a sentence is a list of words. Logo provides tools to manipulate the words of a sentence. In most programming languages, one uses character strings instead of sentences; an English sentence becomes simply a bunch of symbols, some of which are letters and some of which are spaces. But those computer languages don't help the programmer divide this uniform character string into words. Logo's approach is an example of the microworld influence in programming.

sprite: animation of graphics has long been a high priority in Logo implementations, but only recently have small computers gained the power to handle animations well (except, of course, for animations written efficiently in assembly language). Sprites may be thought of as screen objects with a defined appearance (color, shape, size, etc.) and a velocity. Both of these can be changed according to the wishes of the programmer. They "live" on different layers of the screen, as if they are on plastic overlays. Thus, a sprite defined as a picture of an elephant may roam through a forest of trees, passing behind some and in front of others. The Logo programmer does not have to attend to the hiding of lines when one sprite is partially occluded by another—that is handled automatically. In this way, complex animations involving three-dimensional interactions of several screen objects can be as simple as drawing the objects and instructing each on how it should move.

state: the relevant properties of something. For example, the state of the turtle includes its position and the direction in which it's pointing, but doesn't include any of its past history (such as the distance it has traveled). The criterion of relevance here is that the turtle's future behavior, in response to some Logo program, depends only on its current state and not on its past. (Of course, the turtle's state does reflect where it has been, but tells only part of the history and not all of it.) The isolation of only the important aspects of a situation is a valuable debugging tool.

subprocedures: Logo encourages students to deal with large problems by dividing them into subprocedures. In a "long, featureless set of instructions it is hard to see and trap a bug. By working with small parts, however, bugs can be confined and more easily trapped, figured out." (See Papert,

Investigate Robotics

RHINO XR-1
HIGH-TECH ROBOT ■ FOR EDUCATION, RESEARCH & INDUSTRY

built with the same operating technology as large industrial robots

$2400

complete with 32" high robotic arm • power pak • controller card • cables • manual • tool kit

- Use with any RS-232C computer port or modem
- Rigid ¼" open aluminum construction
- Six powerful DC Servo gear motors
- Digital choppers on all axes
- Controller card with Intel® 8748 CPU
- Built-in self test program
- Reach: 22.5" maximum
- Grasp: 2"
- Lift: 16 oz. at extension

"Hands-on-Introduction to ROBOTICS, The Manual for XR-1" may be purchased separately @ $20.00 ea. p.p.

Immediate delivery. Shipped F.O.B. Champaign, IL (IL residents please add 5% sales tax) Price & specifications may be altered without notice.
Dealer/Rep. inquiries invited.

Sandhu Machine Design, Inc., Dept. B-1
308 S. State, Champaign, IL 61820
217/352-8485

Circle 258 on inquiry card.

MODEM

$129.95

No other acoustic modem gives you all these features at this low price.

The MFJ-1232 Acoustic Modem gives you a combination of features, quality and performance that others can't match at this price.

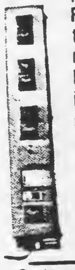
0-300 Baud, Bell 103 compatible. Originate/Answer. Half/full duplex. RS-232, TTL, CMOS level compatible. **Use any computer.** Cassette tape recorder ports save data for reloading or retransmission. 6 pole active filter handles weak signals. Carrier detect LED indicates adequate signal strength for data recognition. Quality "muffs" gives good acoustic coupling, isolates external noise for reliable data transfer. Crystal controlled. "ON" LED. Aluminum cabinet. 110 VAC or 9 volt batteries. 9x1½x4 in.

Apple II, II Plus: software and cable for modem, MFJ-1231, $39.95. Plugs into game port. No serial board needed.

It's like having an extra port **$79.95**

MFJ-1240 RS-232 TRANSFER SWITCH. Switches computer between 2 peripherals (printer, terminal, modem, etc.). Like having extra port. **Push button switches** 10 lines (pins 2,3,4,5,6,8, 11,15,17,20). Change plug or cable to substitute other lines. Push button reverses transmit-receive lines. LEDs monitor pins 2,3,4,5,6,8,20. PC board eliminates wiring, crosstalk, line interference. 3 RS-232 25 pin connectors. 7x2x6 in.

$99.95 **MFJ-1108 AC POWER CENTER.** Adds convenience, prevents data loss, head bounce, equipment damage. **Relay** latches power off during power transients. **Multi-filters** isolate equipment, eliminate interaction, noise, hash. **Varistors** suppress spikes. 3 isolated, switched socket pairs. One unswitched for clock, etc. Lighted power, reset switch. Pop-out fuse. 3 wire, 6 ft. cord. 15A, 125V, 1875 watts. Aluminum case. Black. 18x2¾x2 in. **MFJ-1107, $79.95.** Like 1108 less relay. 8 sockets, 2 unswitched. **Other models available,** write for free specification sheet.



Order from MFJ and try it. If not delighted, return within 30 days for refund (less shipping).

One year unconditional guarantee.

Order yours today. Call toll free 800-647-1800. Charge VISA, MC. Or mail check, money order. Add $4.00 each for shipping and handling.

CALL TOLL FREE . . . 800-647-1800

Call 601-323-5869 in MS, outside continental USA

MFJ ENTERPRISES, INCORPORATED

page 102.) Logo's subprocedures are more than what one sees in carefully subroutinized BASIC. They become extensions to the language and can have meaningful names (see **powerful ideas**). For more on the distinction between subroutines and subprocedures, see the section of Brian Harvey's article on page 163 in this issue of BYTE.

teaching: what a person does to a computer. See **learning**.

Total Turtle Trip Theorem: when a turtle wanders over a path and ends up heading in the same direction as when it started, its total turning is an integer multiple of 360 degrees. Some special cases follow. If it wanders over a simple closed curve (a closed path that does not cross itself, like a polygon, a circle, or the outline of a pond), it turns through exactly 360 degrees. To figure out how much to turn at each point of a five-pointed star, play turtle (remember to end up heading the same way you started). How many full rotations do you make while walking over the star shape? (Two.) Therefore, you have turned 2*360 degrees in all. How much at each corner? (A fifth of that.) Repeating [FORWARD 100 LEFT 200] enough times makes another kind of star. How many points does it have? What about the star made by turning 80 degrees at each corner? See also **turtle geometry**.

turtle: (1) a computer-controlled robot that has position and heading, each of which can be changed independently of the other. It is significant that these two components of the turtle's state are independent. Cars and trucks (and their remote-control toy versions) do not have independent control of heading and motion. Their inability to make sharp corners (no matter how good the turning radius) limits their usefulness in artistic and mathematical realms. Also, because the interaction complicates thinking about their behavior, it is difficult to write programs to control them in any realms; (2) a graphic representation (typically, a pointy isosceles triangle) of such a physical robot; (3) a creature that lots of people love even though the beast rarely does anything in anyone's presence.

turtle geometry: a genuinely new mathematics based on turtle movements. It emphasizes transformations in local space rather than relationships to a fixed global referent. See **local**, **Total Turtle Trip Theorem**, and **playing turtle** in this glossary. For more about the mathematics at a college or advanced high school level, see Abelson and diSessa's book.

turtle graphics: the graphics you know and love—the graphics command system that has crept into several other computer languages (I've seen it in some Pascal implementations and even a version of BASIC) originated with Logo. See **FORWARD**, **Logo**, and **turtle**.

word: the simplest form in which Logo stores data, a word is an unbroken string of characters (typically alphanumerics, but capable of being constructed to contain any characters including spaces or control characters, when that is desired). Words can be concatenated into larger words, dissected into parts, and used as elements of lists. Numbers, single characters, and character strings are all treated as words by Logo.

workspace: a Logo workspace may be thought of quite concretely as if it were a kitchen, basement, or art-studio workspace: a place where one can set out one's materials and begin to work. The Logo workspace contains all the procedures and data objects the programmer is currently using, which can be saved in whole or in part on files for use at a later date. Retrieving the contents of a file places its contents into the workspace, as does defining a new procedure or data object. Pictures or text that are on the screen are not part of the computer's *internal* workspace any more than pictures drawn by a robot turtle on the floor are part of that workspace. (Nevertheless, many implementations of Logo make it possible to save screen graphics on a file and retrieve that file to the screen.)■

The Case Against Pilot:

The Pros and Cons of Computer-Assisted Instruction Languages and Authoring Systems

Paul F. Merrill

The advent of low-cost microcomputers has created a new surge of interest in educational applications in the computer. This new interest is creating a growing demand for quality computer-assisted instruction (CAI) courseware. Most microcomputers which have been introduced in the last few years come with Basic, a general purpose language. Although Basic is considered to be a beginner's language, many feel that it is impractical to expect teachers, with no computer experience, to learn Basic in order to develop CAI courseware.

Since the advent of CAI many individuals and organizations have attempted to develop specialized computer languages which would enable instructors with little computer background to develop CAI courseware. These attempts have fostered such CAI languages as Coursewriter, Planit, Tutor and Pilot.

In an effort to get even further away from computer languages, others have developed authoring systems such as Ticcit and Bell and Howell's recently introduced Courseware Development System for microcomputers.

These authoring systems may be distinguished from CAI languages by their lack of operation or command statements and their use of built-in logic and presentation forms or files. Rather than putting together a series of commands, the author fills in presentation forms such as rules, questions, examples, possible answers, feedback, helps, etc.

Dr. Paul Merrill, College of Education, Instructional Science Dept., W-160 STAD, Brigham Young University, Provo, UT 84602.

CAI languages and authoring systems may simplify the author's task by using one or more of the following techniques: 1) Reduce the domain of possible commands and strategies; 2) Provide commands and strategies which meet the specific needs of instructional applications; 3) Provide commands or routines which perform higher level tasks. Most of the languages and systems mentioned above use a combination of these techniques. However, some languages emphasize one technique over another.

The simplicity of Pilot was achieved by reducing the number of commands in the language.

Pilot

One of the first special purpose CAI languages available for microcomputers was Pilot (Apple Computer Inc. recently announced Pilot for the Apple II). Pilot has gained popularity because it is relatively easy to learn and is designed specifically for educational applications.

The simplicity of Pilot was achieved by reducing the number of commands in the language. Most versions of the language have approximately ten commands. Some of the more powerful versions have several additional commands for graphics, tone generation, and special functions. The principle commands in Pilot include: Type, Accept, Match, Compute, Jump, and Use.

The Type command provides for the display of text and values of numeric and string variables on the screen. The Accept instruction causes the computer to pause and wait for the student to enter a response. The response is edited according to specified rules and stored in a system variable. The response may then be compared with possible correct and incorrect answers supplied by the author using the Match command.

The Compute command provides for the assignment of numeric and string values to variables and the evaluation of arithmetic expressions. The Jump instruction enables the sequence of command execution to be altered by branching to a specified

Table 1. Comparison of Commands.

Pilot	Basic	Pascal
`R: THIS IS A REMARK`	`REM THIS IS A REMARK`	`(*THIS IS A REMARK*)`
`T: GOOD, $N$ , YOU SCORED #T`	`PRINT "GOOD, ";N$;" , YOU SCORED ";T`	`WRITELN ('GOOD, ',NAME,' , YOU SCORED ',T)`
`A: B$`	`INPUT B$`	`READLN (B)`
`M: RED!BLUE&GREEN`	`CA$="RED!BLUE&GREEN": GOSUB 10`	`MATCH ('RED!BLUE&GREEN')`
`J: LABEL`	`GOTO 500`	`GOTO LABEL`
`JY: LABEL`	`IF Y THEN 500`	`IF Y THEN ---`
`TN: SORRY, THE ANSWER IS $A$`	`IF N THEN PRINT "SORRY, THE ANSWER IS ";A$`	`IF N THEN WRITELN ('SORRY, THE ANSWER IS ',A)`
`U(T>13): SUBLABEL`	`IF T>13 THEN GOSUB 800`	`IF T>13 THEN PROCEDURENAME`
`E:`	`RETURN`	`END`
`C: X=4*B+3`	`X = 4*B+3`	`X:=4*B+3`
`D: W(20,15)`	`DIM W(20,15)`	`TYPE W=ARRAY [1..20,1..15] OF REAL`

location in the program. The Use command provides for the execution of subroutines.

When writing a Pilot program the command names described above are abbreviated to one letter (See the first column of Table 1 for examples). These commands may be followed by a character or an expression which serves as a modifier or conditioner of the command. A modifier character changes the nature of the command. For example, an "S" following the Match command allows for one-letter misspelling when comparing the student's answer to the author's answer. A conditioner character or expression causes the command to be skipped unless the specified condition is met. For example, a Type command followed by a "Y" will only be executed if the previous match was successful (the student's answer matched the author's answer).

The command and optional modifier and/or conditioner must be followed by a

Kelly's Computing

ATARI SPECIALS

CIRCLE 169 ON READER SERVICE CARD

colon. The colon may be followed by appropriate parameters such as the author's answers to be matched, the expression to be evaluated, etc. Examples of these features may be found in the first column of Table 1.

A CAI language such as Pilot which tries to simplify the courseware author's task by reducing the number of commands in the language has a significant liability. The restriction in the domain of commands creates a restriction in the range of possible outcomes or applications.

Several versions of Pilot written in other high level languages such as Basic expanded the domain of the language by allowing instructions from the host language to be included in a Pilot program by embedding them as parameters into Pilot Compute commands. The severe liability inherent in the limited Pilot commands became very evident to the author while reviewing the code for a Pilot program where almost one-half of the program statements were Compute commands which used instructions from the host language.

The original version of Pilot developed at the San Francisco Medical Center had such a restriction of domain and range that many institutions began developing extensions to the language. Attempts have been made to standardize these extended features into a version called Common Pilot. However, many of the extended features of Common Pilot are merely adaptations of commands available in other high level languages. The Apple version of Pilot has been further extended to include graphics and sound editors.

The principle reason for selecting Pilot over other, more powerful, general purpose languages such as Basic or Pascal seems to be the ease with which it can be learned. However, this reasoning may be somewhat fallacious. Table 1 lists the principle commands available in Pilot along with corresponding commands in Basic and Pascal which perform similar functions. This comparison reveals that a subset of Basic or Pascal which matches the domain of Pilot commands could be learned just as easily. However, the more powerful languages offer the advantage of additional capabilities when the author is ready to go beyond the minimal subset. If an author begins by learning Pilot and then desires greater power, he must then scrap Pilot and begin learning a new language. Why not begin with a powerful language in the first place?

Although Table 1 shows that a subset of Pascal would be just as easy to learn as Basic or Pilot, a word of caution is in order. Most textbooks and manuals on Pascal currently available do not begin with a simple subset of Pascal that can be easily learned. Rather than beginning by showing how commands can be used in a simple, straightforward fashion, most authors try to present commands in their total complexity. The Pascal language is also embedded in a powerful but complex operating system which includes a text editor and file handler. In order to program in Pascal successfully, an author must also learn how to use the Pascal operating system.

The very nature of the principal Pilot commands strongly encourages novice programmers to use a mediocre strategy.

In addition to simplifying the author's task by reducing the domain of commands, Pilot also includes commands designed specifically for instructional applications. This technique is clearly laudable and is obviously necessary in any effort to tailor a computer language for courseware development. However, when this approach is coupled with a restriction in the domain of commands, additional liabilities are incurred. The majority of the commands may be tailored to specific instructional needs while more general purpose commands are excluded. Often the specific commands selected are chosen based on an implied instructional strategy. The availability of this set of commands and the exclusion of more general commands inadvertently requires authors to use a particular strategy and further limits the range of possible applications.

Several CAI languages, including Pilot, include a set of commands which foster a tutorial or dialogue CAI instructional strategy. A tutorial strategy includes a repetitive sequence of the instructional elements or frames listed in Table 2. This particular strategy is advocated by some because of its similarity to a strategy that might be used by an instructor tutoring a student one-on-one. If the segment to which a program branches is made conditional on the student's response then the instruction can be customized to the needs of each student.

Although such a strategy sounds great, in practice it is generally just plain boring. Often the questions asked merely require the student to copy words or data presented in the previous information frame. In fact, one could argue that the resulting CAI program is little more than a fancy and expensive page turner. The same strategy could be successfully implemented using a branching program text.

Obviously Pilot does not force an author to use a poor tutorial strategy. It is possible to implement other types of strategies using Pilot. One can also develop more creative tutorial programs. However, the very nature of principal Pilot commands strongly encourages novice programmers to use a mediocre strategy.

One of the main advantages of Pilot is the provision of the high level command Match. The Match command allows a Pilot program to perform moderately sophisticated natural language processing. Students may enter free form answers which can be successfully analyzed by the Match command. Many general purpose languages such as Basic, APL and Pascal do not have a comparable command. The author must write special subroutines or procedures to perform similar functions. (See Table 1.)

All computer languages provide a set of basic or primitive commands which can be used to define and perform a higher order function. A set of computer commands organized to perform a specific function is often referred to as a program. Most languages also allow such a set of commands to be defined as a subroutine which can be called from different places in a main program.

These subroutines can subsequently be used as higher order commands to perform complex functions which cannot be performed by a single primitive command in the language. Such higher order commands can greatly simplify the author's task while simultaneously providing greater power.

Although Pilot and Basic provide for

Table 2.

Frames	Pilot Commands
1. Present information	T: Information
2. Ask questions	T: Question
3. Accept student response	A:
4. Check for correct answer	M: Correct Answers
5. Give appropriate feedback	TY: Very Good
	TN: Sorry, that's wrong
6. Branch to next segment	JY: Next

the specification and calling of subroutines, this capability is severely limited. Many versions of Basic refer to subroutines by number, which reduces the readability of the program.

Neither Basic nor Pilot provides for the passing of parameters from the main program to the subroutine. This makes it nearly impossible to develop subroutines which are truly modular. They cannot be used as a black box and plugged into any program without revision. Thus they are not very useful as higher order commands to an author who is not familiar with their internal workings. The variables in the subroutines are not local to the subroutines and thus may affect variables elsewhere in the main program. This can create bugs in the program which are difficult to find and correct.

On the other hand languages such as Pascal, APL, Logo and the new Actor languages such as Smalltalk provide for truly modular subroutines. Variables can be made local to a subroutine. Once such subroutines have been developed and verified to perform a function, they can be used as higher order commands without regard to their internal operations.

Pascal even allows for such subroutines to be placed in a system library and subsequently linked into any program with a single command. Such capabilities make it possible to extend a language and tailor it to meet specific needs. As mentioned earlier, defining higher order commands in this way simplifies the author's task while at the same time providing greater capacity.

Authoring Systems

Authoring Systems have been developed in an attempt to overcome the problem of having to learn a computer programming language. This is generally accomplished by separating the logical sequencing and the control of the computer program from the instructional content. The logical sequencing of instructional frames or the instructional strategy is preprogrammed into the authoring system. The author is therefore relieved from worrying about the logic and strategy of the courseware and may concentrate on the content of the instructional frames. The authoring system provides the author with a template in which to place the content to be presented.

For example, in the Bell and Howell's Genis Courseware Development System the author merely enters the text for presentation frames, question frames, correct answers, incorrect answers, feedback messages for each alternate answer, etc. The author is prompted by Genis to enter each item of information. The sequence and instructional logic is automatically built into the system.

Authoring in the Ticcit system is slightly more complicated, but the resulting courseware is more flexible. Ticcit was designed based on the philosophy of learner control. When students interact with a completed Ticcit lesson, they are given control over the content and sequence of the instructional segments they study.

The desired presentation form is selected when the student presses special learner control keys on the keyboard. These keys are labeled: Objective, Rule, Example, Practice, Help, Map, Advice, etc. Thus, if students press the key labeled Practice, they will receive a practice item. If they press Advice, they will receive guidance on what to do next. The map provides a table of contents in diagram form showing the hierarchical relationship between lessons. Students may select a different lesson segment when the map is displayed.

Authoring systems reduce cost and effort by reducing variety in much the same way that cost and effort are reduced in fast food restaurants.

Ticcit provides several packaging routines which allow an author to type in content for the rules, examples, practice items, etc. that will make up a lesson. As with Genis, the Ticcit system automatically takes care of the storage and retrieval of the various presentation form files. Thus the author does not have to worry about the computer logic required to display the appropriate instructional frame or presentation form when students press one of the learner control keys.

Authoring systems such as Genis and Ticcit simplify the author's task by using all three techniques listed earlier: very high level routines are built into the system to control the logical sequencing of the instruction; this logical sequencing is designed according to an instructional strategy which meets the specific needs of certain instructional applications; and the domain of possible strategies is obviously reduced.

Authoring systems reduce cost and effort by reducing variety in much the same way that cost and effort are reduced in fast food restaurants, tract homes and formula television shows.

With Ticcit and Bell and Howell's Courseware Development System the author is given a template to use. Although the template greatly simplifies the task, it forces the author to turn out courseware which conforms to the template. Such templates can have the effect of enhancing the quality of the courseware produced by the novice, while restricting the quality of the courseware produced by a creative author. The general quality of the courseware produced is dependent on the quality of the template.

Conclusions

•Authoring systems or languages allow authors to turn out respectable courseware using a specific template. However, authors must recognize the limits of the template, and not try to force all instruction to fit the template. The use of templates would be more acceptable if a variety of templates were provided for the different types of learning.

•CAI languages should not reduce the domain of commands. Beginning authors should initally be taught a simple subset of a language which will allow them to begin writing simple programs quickly. However, they should then have the option to learn additional commands which will give them additional capabilities and allow them to develop more sophisticated courseware.

•CAI languages should not be restricted to commands developed specifically for certain types of instructional applications.

•Experience in developing CAI courseware and in training students to develop courseware has shown that authors very quickly reach the limitations of whatever language they are using. They then clamor for additional capabilities.

•CAI languages should provide for the straightforward development and use of higher order commands which increase the power of the language but do not exclude elementary operations.

•The best language might be a general purpose language, which provides great flexibility, with the provision of higher order commands for instructional applications.

•When possible, subject matter experts with little computer experience should team up with programmers who have had considerable experience programming in a sophisticated language.

As microcomputers invade the home and classrooms the need for large quantities of quality courseware will become acute. It is imperative that authors select an appropriate tool to maximize their productivity and the quality of their courseware products. Care must be exercised to avoid selecting tools which will sacrifice quality for quantity. □